Bruno Miguel Ourives Nunes

Licenciado em Ciências da Engenharia
Eletrotécnica e de Computadores

The Design and Testing of a Picosatellite, The TubeSatellite CTSAT-1

Dissertação para obtenção do Grau de Mestre em

Engenharia Eletrotécnica e de Computadores

Orientador:	Prof. Doutor João Pedro Abreu de Oliveira, Professor Auxiliar, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Júri

Presidente:	Prof. Doutor Rui Miguel Henriques Dias Morgado Dinis - FCT/UNL
Arguente:	Prof. Doutor Nuno Filipe Silva Veríssimo Paulino - FCT/UNL
Vogal:	Prof. Doutor João Pedro Abreu de Oliveira - FCT/UNL

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2019

The Design and Testing of a Picosatellite,
The TubeSatellite CTSAT-1

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "novathesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

À minha família.

Acknowledgements

I would like to acknowledge my supervisor Prof. João Oliveira for his guidance during the development of this project and writing of this thesis.

To Ricardo Madeira and Nuno Correia from Centre of Technology and Systems of Uninova for their tips and guidances on SMD soldering and PSoC communications implementations.

Finally, to my family for their unlimited support and encouragement, which was vital to accomplish this milestone.

Abstract

This thesis consists in designing and testing the payload of a TubeSat low-cost satellite, called CTSAT-1. Instead of spend large quantities of money to space research, this picosatellite offers a very low-cost solution to provide answers for institutions investigations.

Designing and assembling a picosatellite of this size represents a big challenge due to packing of a lot electronics and equipments inside TubeSat low dimensions, therefore providing a unique efforts in reaching a complete small spacecraft.

To keep the budget of the project low as possible, maintaining TubeSat specifications and space flight requirements, for intensive testing will be used homemade techniques to achieve satisfactory results without spending too much on expensive equipment.

Keywords: TubeSat, picosatellite, low-cost satellite, design, experimental testing

Resumo

Esta dissertação consiste em projetar e testar a carga de um TubeSat, satélite de baixo custo, designado CTSAT-1. Em vez de se gastar grandes quantidades de dinheiro para a pesquisa espacial, este picosatélite oferece uma solução de baixo custo para fornecer respostas para investigações institucionais.

Projetar e montar um picosatélite deste tamanho representa um grande desafio devido ao empacotamento de muitas peças eletrónicas e equipamentos dentro das dimensões reduzidas de um TubeSat, fornecendo consequentemente um esforço único em alcançar um aparelho espacial pequeno.

Para manter o orçamento do projeto o mais baixo possível, mantendo as especificações do TubeSat e as exigências de voo espacial, para testes experimentais intensivos serão utilizadas técnicas caseiras para obter resultados satisfatórios sem gastar muito em equipamentos caros.

Palavras-chave: TubeSat, picosatélite, satélite de baixo custo, projecto, testes experimentais

Contents

List of Figures

List of Tables

Listings

Acronyms

ARM	Advanced RISC Machine.
ELF	Extremely Low Frequency.
ESA	European Space Agency.
ESTEC	European Space Research and Technology Centre.
IOS	Interorbital Systems.
IPMA	Instituto Português do Mar e da Atmosfera.
ISS	International Space Station.
LEAF	Large European Acoustic Facility.
LEO	Low Earth Orbit.
NASA	National Aeronautics and Space Administration.
PCB	Printed Circuit Board.
P-POD	Poly-Picosatellite Orbital Deployer.
PSoC	Programmable System-on-Chip.
RATF	Reverberant Acoustic Test Facility.
RTC	Real-Time Clock.
SAA	South Atlantic Anomaly.
SPI	Serial Peripheral Interface.
SSDL	Space Systems Development Laboratory.
USB	Universal Serial Bus.

Chapter 1

Introduction

1.1 Motivation

For a long time mankind has always wanted to go beyond the surface of the Earth. Missions such as Sputnik or Apollo have become a reference for space exploration. In recent years, due to the dizzying evolution of technology, it is possible to achieve new goals using fewer resources.

The use of small and low-cost satellites has been increasing, which translates into an opportunity for students and researchers to develop missions with lower budgets.

This thesis will evaluate one of the most important phases in a space project, the payload development process. It will also contribute to a solid and ambitious project that is to build a small space device.

According Interorbital Systems (IOS), Universidade Nova de Lisboa is the only portuguese university belonging to the launch manifest of TubeSats [30]. Considering that is listed only one TubeSat kit for this university, that means this project, at the time of this writing, is the only portuguese TubeSat.

1.2 Thesis Organization

This thesis is presented into six chapters, including the present introduction, the remainder can be summarized as follows.

- **Chapter 2 : State of the Art** : in this chapter is presented a bibliographical review of the work about space related concepts. Is introduced the concepts of space environment, low-cost satellites and its applications. There is also information about testing technics of satellites.

- **Chapter 3 : CTSAT-1 - Design and Implementation** : in the third chapter, there is the description of the low-cost satellite used including its structure and main stages. Also is described a detailed implementation of the payload system.
- **Chapter 4 : Testing Procedures** : this chapter includes the testing methods that are used to test the TubeSat.
- **Chapter 5 : Measurement Results and Analysis** : it is a chapter that includes withdrawn results from the testing phase. After that, the results are analysed.
- **Chapter 6 : Conclusions and Future Work** : here are the final considerations about the work, with a summary of the main conclusions to be drawn. It also has prospects for future work in order to improve the work done.

Chapter 2

State of the Art

2.1 Orbital Conditions

Some satellites usually operate in the Low Earth Orbit (LEO) that is located at an altitude between 150 km and 600 km. This is the region that also has the the International Space Station (ISS) and several science and communication satellites.

LEO is in the part of the atmosphere called the Thermosphere, and the Thermosphere contains the region called the Ionosphere, layer that is composed mainly of electrically charged atoms (ions) and electrons, due to the ultraviolet radiation from the Sun, that also coincides with some of the Earth's magnetic field.

The Earth's magnetic field shields us from the Sun's most activity. High-energy particles, flare emissions, and coronal mass ejections get shunted by the magnetic field before they can reach ground. Where the magnetic field lines dip near the poles, this energy expresses itself as the aurora (Figure 2.1) [13].

Above the ionosphere, the space environment can be hostile because of solar activity. Below it, the radiation risks are much lower thats why most satellites are kept in LEO region. Below the altitude of 150 km it is impossible to maintain a stable orbit.

A typical LEO orbit has a period of 90 minutes, in other words, an object inside LEO rotates around the Earth once every 90 minutes, doing about 16 orbits per day, each time with position shifting a bit.

Figure 2.1: Low Earth orbit view of an aurora by ISS [18].

2.1.1 Temperature

The temperature in LEO can vary between −170°C and 123°C [15], depending on the rotation speed of the satellite and the time that the satellite passes towards the Sun.

Since the satellite it is spinning around itself, due to rocket satellite launch mechanism, this range is fortunately smaller, which means that the heat has time to distribute and dissipate. An complete orbit has approximately half its time in sunlight and the other half in Earth shade [13].

Some sensors are only guaranteed to −40°C, since most electronics have trouble below −40°C, so if in any event, past below that range, the rest of the satellite electronics may have trouble.

Due to that temperatura range, in the 90 minute orbit, it is possible to cycle through three ranges, one is the sensor is too cold to register, other is a transition region where the sensor returns valid, slowly changing data, and the other is a possibly over saturating at the high end [13].

Even so, there is some on-board heating of satellite components due to being in use, these can achieve a temperature range of −30°C to 90°C. Those values are often the limits from the off-the-shelf electronics components. The design of a satellite must have take into account this factor to keep the temperature controlled to work properly.

2.1.2 Luminosity

The light coming from the Sun, could be used to measure its level of intensity, but since the spiral movement of the satellite, it could only measure the timing of when the

Sun is in view, because most function of the light sensors will return a binary value and no sensor will be able to register the value off the luminosity when it is directed directly to the Sun.

Therefore these light sensors will provide a basic measure the satellite's movement cycles and also record the overall day/night cycles during the period in which it is operating.

If the mission objective was to measure actual light levels, there was necessary a special detectors design ensuring the Sun doesn't saturate the light detector.

2.1.3 Magnetic Field

The ionosphere has a magnetic field strength between 0.3 - 0.6 Gauss, with variations of 5% [13].

A common Hall effect sensor, which is based on the Hall effect, tends to be designed for Earth work and could measure tens of gauss, so for satellites it is need to ensure that the sensor is calibrated for the space environment, not Earth's surface.

Because of the background noise generated by internal magnetic field of the Hall effect sensor, this sensor it is not capable to measure fields with an intensity less than 0.63 Gauss.

However, a Hall effect sensor plus an op-amp could measure variations down to as low as 0.06 Gauss if there's no large external magnetic field [15].

2.1.4 Outgassing

The Outgassing or sublimation, is the vaporization of a solid going to a gas state due to low pressure.

With values of pressure of 10^{-11} to 10^{-15} Pa of LEO, many materials that are stable in the Earth surface will outgas in space. This process occurs at an increasing rate as temperature rises.

This phenomenon has two problems, it can erode the material that is outgassing, but the outgassed material may condense on and then coat other surfaces. This can change their conductance for some components or, for detectores and optics, coat them so they no longer work properly.

For this reason, lubricants used on the ground are clearly not appropriate to space operation, because lubricants have a high vapor pressure and outgas quickly, which makes them to evaporate off moving parts and they can coat near surfaces [23].

The atmospheric composition indicates that in LEO has about 96 % atomic oxygen. In the early days of National Aeronautics and Space Administration (NASA) space shuttle missions, the presence of atomic oxygen caused problems, so it became apparent that atomic oxygen provides an aggressive environment for materials used on space vehicles in LEO [17].

2.1.5 Space Debris

Since the launch of Sputnik I in 1957 the space activities have created an orbital debris environment that create an increasing impact risks to existing space systems.

The major source of the current population of debris objects larger than about 1 cm are fragmentations. Most causes for these fragmentation are aerodynamic, deliberate, explosion, propulsion related, electrical and collision [7].

The current debris population in the LEO region has reached the point where the environment is unstable and collisions will become the most dominant way to generate more debris in the future. Even without new launches, over the next 200 years collisions will continue to occur in the LEO environment [33], but in reality, the situation will be worse because spacecraft objects will continue to be launched.

Capturing space debris is a problem that some institutions are trying to solve, because it's not just launch a rocket and command it to fly faster, cause any change in speed also changes rocket's altitude.

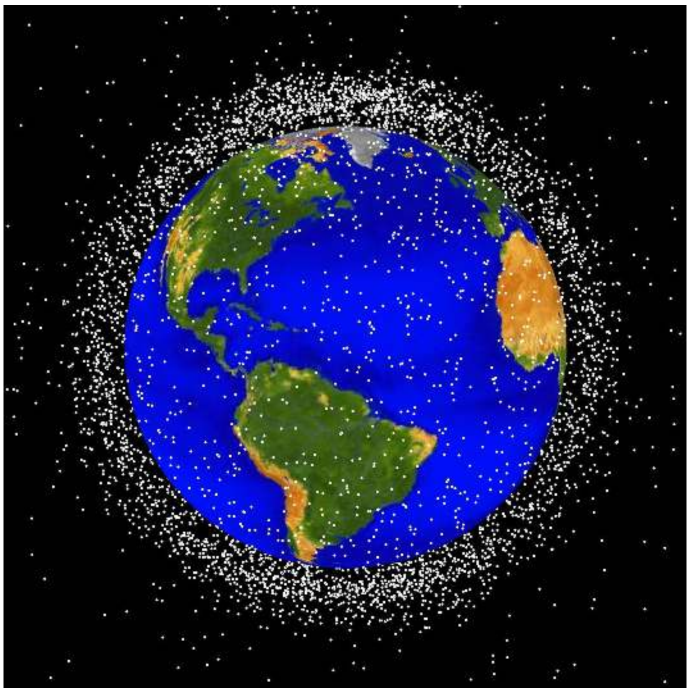


Figure 2.2: Simulated image showing debris in LEO [52].

2.1.6 Radiation

In space, the radiation originated by solar activity can damage satellites. The primary source of damage is due to highly energetic electrons, protons, and ions emitted by the Sun.

The shape of the Earth's magnetic field also can lead to dips in the field that cause a higher hazard in certain geographic areas, like near Brasil, where is the South Atlantic Anomaly (SAA), that is a region where the Earth's magnetic field lines dip lower and thus increase the electron/proton flux experienced in orbit and exposes orbiting satellites to higher than usual levels of radiation. These charged particles can cause temporary and permanent damage to satellites in space, causing single event upsets that make random signals appear in electronics of the satellite. LEO satellites can pass through during one or more orbits each day and satellites with sensitive detectors often decide to shut down or cease collecting data during the passage on this location [26].

Particles of low energy will be shielded by the body of the picosatellite (Figure 2.3).

The particles with very high energy will be slowed by the small satellite and will typically pass straight through the satellite and electronics (Figure 2.4). However, there is a probability of some of this particles penetrate inside the satellite and deposit some energy on the components.

The damaging particles are the particles with a specific energy. They have enough energy to pass through the satellite's surface to get into the electronics, but not so much energy to pass through the satellite. Instead, they deposit their energetic charge into the electronics, causing damage or single event upsets (Figure 2.5). It can also degrade the solar panels and other sensitive components [14].

Shielding can help protect electronics, but can also increase the risk of damage, because shielding attenuates all particles. These means that the low energetic particles are slowed down so much they are completely blocked. The particles with right amount of energy that without shielding could cause damage, with shielding they can be blocked by satellite's surface. The particles that have high energy, they lose some energy passing through the shielding, but still continue to move to inside the satellite leading to possibly cause more damage.

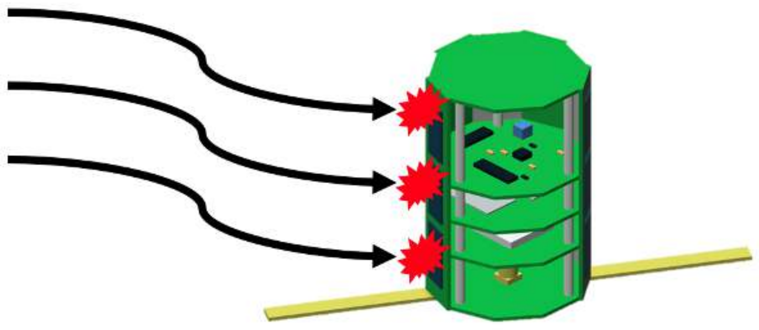


Figure 2.3: Effects on a picosatellite of low energetic particles.

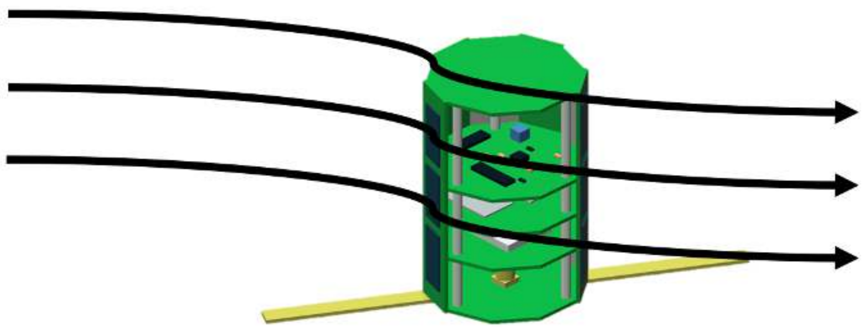


Figure 2.4: Effects on a picosatellite of high energetic particles.

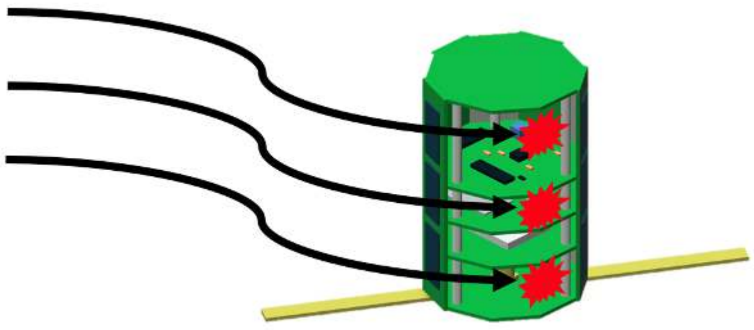


Figure 2.5: Effects on a picosatellite of particles with the right amount of energy.

2.2 Low-Cost Satellites

2.2.1 Miniaturized Satellites

Until recent years, the space exploration were limited to big corporations with high budget. These corporations are responsible for creating heavier satellites that require larger rockets with greater thrust and very complex and expensive equipment to perform space missions.

Miniaturized satellites are small satellites with low weights and small sizes, usually under 500 kg. While such satellites can be referred just as small satellites, different classifications are used to categorize them based on mass.

Definition	**Size**
Minisatellite	100 - 500 kg
Microsatellite	10 - 100 kg
Nanosatellite	1 - 10 kg
Picosatellite	0.1 - 1 kg
Femtosatellite	0.01 - 0.1 kg

Table 2.1: Classifications that are used to categorize miniaturized satellites based on mass [31].

The main reason for miniaturizing satellites is to reduce the cost of development and launch to provide researchers and even students an equipment for making experiments on space. Smaller and lighter satellites require smaller and cheaper launch vehicles, and are often suitable for launch in multiples.

One advantage of picosatellites is after operating for a several weeks or months, they will safely re-enter the atmosphere and burn-up not contributing to the long-term build-up of orbital debris.

But small satellites also have some technical challenges as they usually require innovative propulsion, attitude control, communication and computation systems.

The first launch of a picosatellite was realized on 30 June 2003 with the launch of 6 CubeSats [22]. Since then the market of miniaturized satellites is registering a fast growth.

Picosatellites must be powered off at the time of delivery until their deployment in orbit. Therefore, they cannot radiate radio frequency emission, either as radio signal or electronics noise.

2.2.2 CanSat

The CanSat is a representation of a real satellite, integrated within the volume and shape of a soft drink can (330 ml). Since the maximum mass allowed of the CanSat is limited to 350 grams, it can be considered as a picosatellite.

CanSats offer a opportunity for students to have a first practical experience of a real space projects. They are responsible for selecting its mission, designing the CanSat, integrating the components, programming the on-board computer, testing, preparing for launch and then analysing the data.

There are some importante rules to to accomplish like the CanSat should have a recovery system, such as a parachute and the total budget of the model should not exceed 500 €. This device must be powered by a battery and/or solar panels, because explosives, detonators, pyrotechnics, and inflammable or dangerous materials are strictly forbidden [60].

Unlike other picosatellites, the CanSat operates in the troposphere, which means that will never reach orbit. This layer contains about 80% of the total mass of the atmosphere, and stretches to about 10 kilometres altitude [10].

The CanSat is launched by a rocket to a previously defined altitude (approximately 1000 meters) so that during the descent it is possible to carry out a scientific experiment, capture the signals emitted and ensure a safe landing.

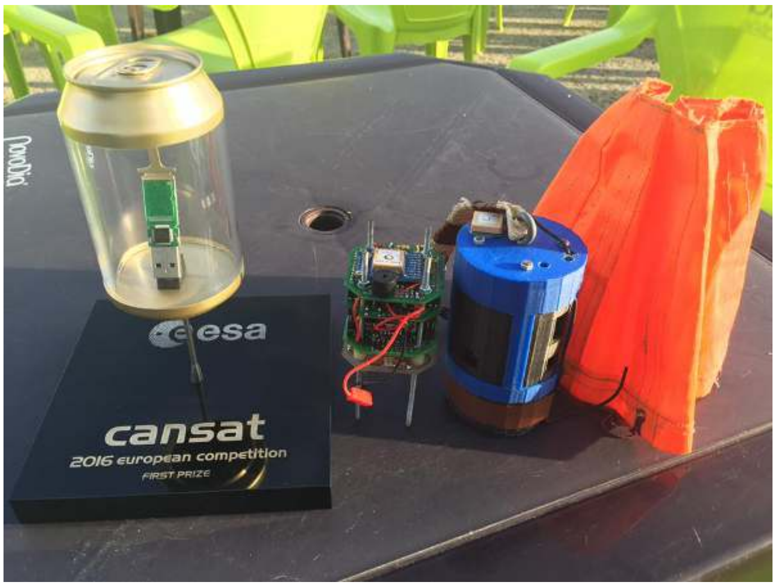


Figure 2.6: Portuguese ENTA team SAT 2, winner of the 2016 ESA's European CanSat competition [24].

2.2.3 CubeSat

The CubeSat is a type of miniaturized satellite belonging to the genre of picosatellites used primarily by universities for space exploration and research, typically in low Earth orbit.

The design standard was created in 1999 by two university professors Jordi Puig-Suari and Bob Twiggs of the universities of California Polytechnic State University and Stanford University, respectively. The main purpose of the project is to provide a standard for design of picosatellites to reduce cost and development time, increase accessibility to space, and sustain frequent launches to universities and educational institutions, but also private firms and government organizations. A typical CubeSat launch cost is estimated at 40,000 $ [13].

The picosatellite CubeSat 1U is a 10 cm cube with a mass up to 1.0 kg [4]. It is possible to group several identical devices in order to increase the size and number of necessary components to carry out the different missions, such as, the CubeSat 2U which measures 10 x 10 x 20 cm.

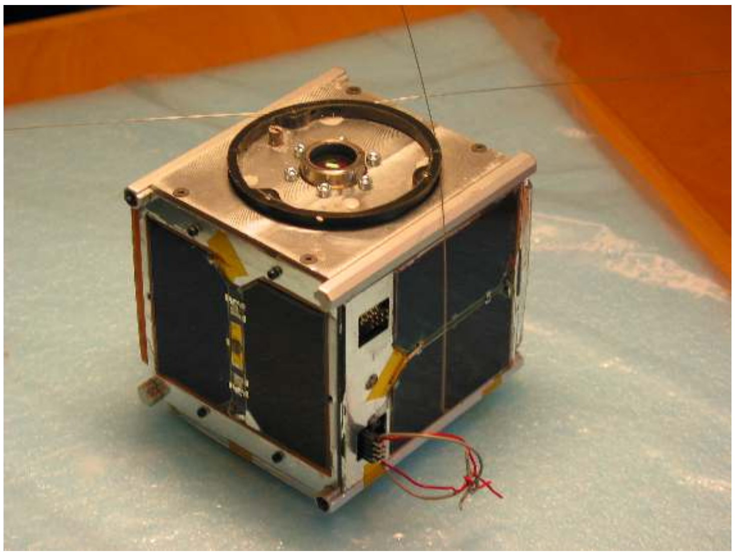


Figure 2.7: AAUSat, Cubesat 1U developed by students of Aalborg University [1].

The constitution of CubeSat is based on multi-layers design: the antenna, the controller layer, the communication layer, the power management layer and the payload area. The structure is made with an aluminum frame.

This kind of satellites are launched and deployed using a common deployment system called a Poly-Picosatellite Orbital Deployer (P-POD) developed and built by California Polytechnic State University.

Only few of CubeSats are equipped with a propulsion system that enables orbit correction or attitude control, such as, the CubeSat built by the University of Illinois, which was loaded with an array of small ion thrusters [31].

2.2.4 TubeSat

The TubeSat is a hexadecagon shaped picosatellite and is the low-cost alternative to the CubeSat and it was designed by IOS, which sells a kit, including some main hardware components and launch for the price of 8,000 $, which is much lower than the CubeSats costs. The first TubeSat kits reached the market in 2009 [29].

This picosatellite has 12 cm in length with a mass up to 0.75 kg, but maintaining the Cubesat volume. As CubeSats, the TubeSats can also group several identical devices to meet the necessary mission.

They are designed to operate for up to 2 months, depending on the solar activity verified during the orbit. After that they are placed into self-decaying orbits 310 kilometers above the Earth's surface [58].

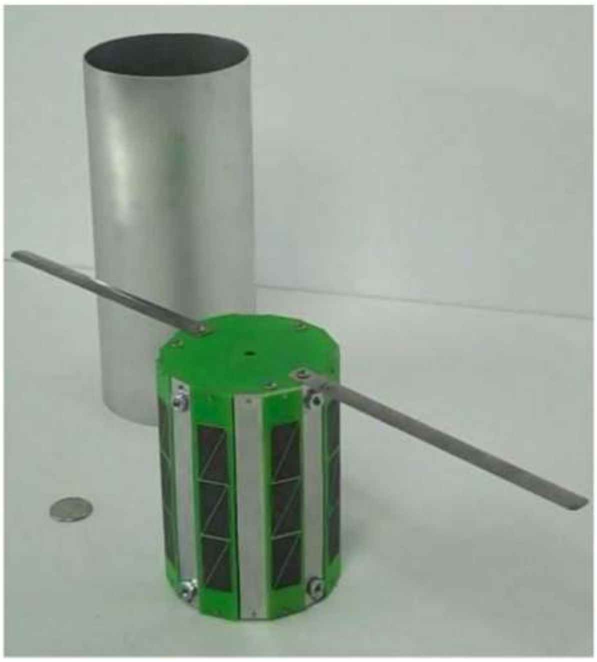


Figure 2.8: TubeSat with a sample ejection cylinder [59].

This picosatellite is also based on multi-layers design but it's structure is assembled from a set of printed circuit boards (PCBs), instead of having an aluminum frame, like CubeSats.

The TubeSats are launched into orbit on an IOS Neptune three-stage modular rocket. These rockets designed to place between 30 and 1,000 kg into low Earth orbit. The 30 kg payload capacity allows formations of 24 TubeSats to be launched per orbital mission with each TubeSat in its own dedicated deployment unit [58].

The first assembled TubeSats were launched in late January, 2017, with the TANCREDO-1 and OSNSAT being the first picosatellites of this type to reach LEO [29].

2.2.5 Picosatellite Constellations

With the increase of projects using low-cost satellites, the complexity of the objectives has also increased, making it clear that it would be impossible to developing missions using only one equipment. Therefore constellations of picosatellites could prove to be a low-cost and efficient solution to carry the proposed missions.

The benefits of several satellites in different orbits working in a network are the increased temporal and spatial resolution of observations and measurements. Additionally, with a larger number of less-expensive satellites involved, a mission is less affected by single-satellite failures.

There are several common constellation architectures [36]:

- Geosynchronous - Worldwide coverage using three to five satellites.
- Ellipso - Several satellites with an elliptical orbits.
- Polar Non-symmetric - Satellites in polar orbits with varying rotational spacing.
- Walker/Rosette - Satellites in individual rotationally symmetric orbital planes with identical altitudes and inclinations.
- String of Pearls - Multiple satellites in the same orbit plane.
- Streets of Coverage - polar orbits with satellite right ascensions of ascending node (RAAN) spread evenly across one hemisphere

Through these constellations, it is possible to increase the complexity of the missions, since there is the possibility of optimize coverage over a specific region to improve the measurements in certain applications, such as Earth observation, communications, remote sensing or reconnaissance.

One of the projects using picosatellite constellations is the QB50 that has the objective to demonstrate the possibility of launching a network of 50 CubeSats built by universities teams all over the world, to perform atmospheric research within the lower thermosphere, between 200 - 380 km altitude [41].

2.3 Picosatellite Application Areas

With the accessibility of low-cost satellites, there is several fields with projects that enable a vast array of research possibilities and applications.

Some of fields are education, earthquake detection, Earth observation, military, biology, communication, technology demonstration, propulsion experiments, events monitoring, radio, management of disasters, environmental, re-entry material testing, inspection of larger satellites, navigation, scientific research between many others [21].

In the following subsections there are some information about some of fields mentioned earlier.

2.3.1 Earth Observation

The mission of the AUUSat (Figure 2.7) was to photograph several areas around the globe from the LEO.

To accomplish this objective , this CubeSat satellite was equipped with Kodac CMOS image sensor that provided a resolution of 1280x1024 pixels (1.3 megapixels) with a color depth of 24bit colors. From a height of 900 km this optics provide an on ground resolution of approximately 150x120 m. The structure in which the lenses were placed were made from Titanium [8].

Once in orbit, this satellite started to have failures due to battery problems.

Flock-1 is a constellation of CubeSats started to be launched from the ISS in January 2014. This constellation is developed by Planet Labs and initially consisted in 28 satellites capable of providing medium to high resolution imaging of the entire planet with a ground resolution of 3 to 5 m.

Most of the Flock 1 satellites were RGB imaging systems, but some containing different optical spectral bands and camera firmware [5].

After the launch of these 28 satellites, Planet Labs still continue to launch new Flock-1 satellites.

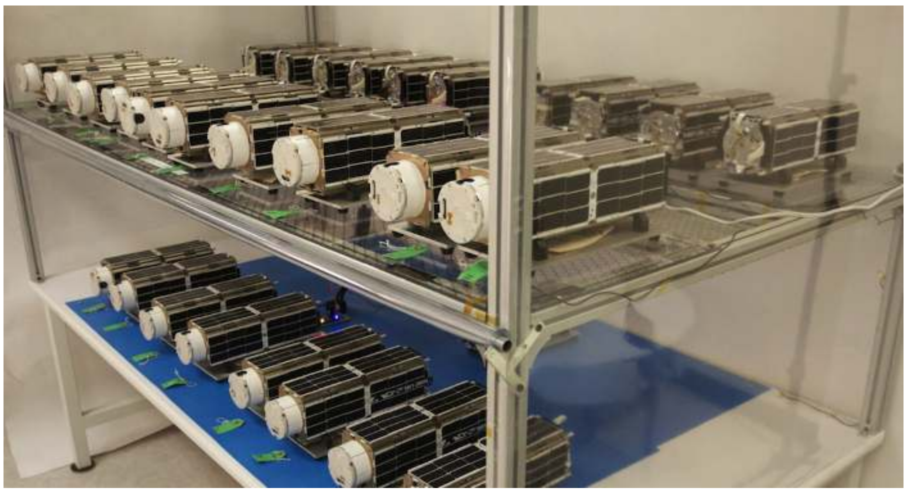


Figure 2.9: Twenty-eight satellites from Planet Labs ready for launch [2].

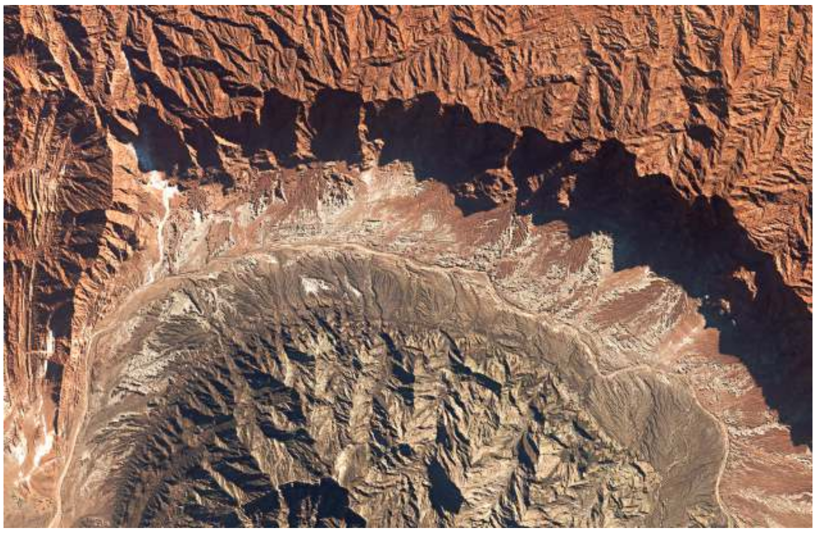


Figure 2.10: Valle de la Luna in Argentina. Image acquired on 19 July 2016 by a recent Flock-1 satellite [48].

2.3.2 Earthquake Detection

The QuakeSat was a miniturized satellite that is size of a triple CubeSat developed by the Space Systems Development Laboratory (SSDL) at Stanford University and the QuakeFinder Team. It was launched on June, 2003 last more than a year in orbit. It carried a very sensitive magnetometer.

His mission objectives were to detect, record, and downlink the ELF magnetic data to verify the results. In addition, it was hoped that investigating the ELF data can ultimately lead to the prediction of earthquakes of magnitude 6.0 or higher.

During an earthquake, theory states that fracturing of the rock creates the Extremely Low Frequency (ELF) magnetic waves. These signatures radiate from the earthquake hypocenter region, through the earth to the Ionosphere, and are propagated up the Earth's magnetic field lines to the satellite altitude (600-900 km) [9].

QuakeSat records ELF magnetic field data in the 1-1000 Hz range in 4 bands, one band at a time [11].

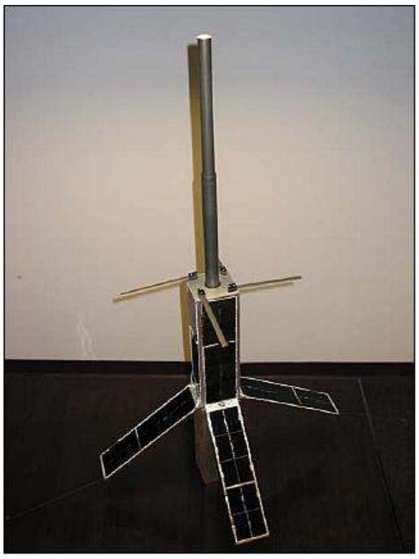


Figure 2.11: Assembled QuakeSat satellite [3].

2.3.3 Biology

PharmaSat was small satellite developed by NASA's Ames Research Center and Santa Clara University and it was launched on May 2009. It has the constitution of 3 CubeSats 1U, meaning that is a 3U CubeSat.

The mission of the PharmaSat was to investigate the efficacy of anti-fungal agents in the spaceflight environment [6].

The satellite contains a controlled environment micro laboratory with sensors and optical systems that can perform the necessary experiences and transmit that data to scientists for analysis on Earth. PharmaSat is equipped with some subsystems [45]:

- An optical sensor system that is able to detect the growth, the health and size of the yeast populations;
- A miniature environmental controller to monitor the experiences;
- A micro-fluids system consisting of 48 small holes, which house four different sample groups of yeast, and a network of small tubes to feed the yeast with a sugar solution and dose them with an anti-fungal agent.

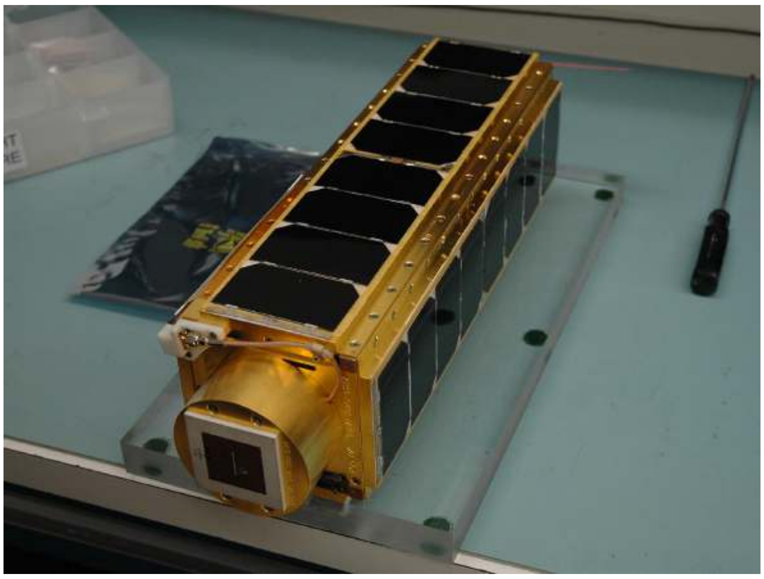


Figure 2.12: PharmaSat ready for testing [44].

2.4 Testing Environment

One of the most important phases of a technological project is the testing phase. In the space industry this phase is extremely important because physical access to equipment is not possible while it is in operation.

A rocket that provides satellite's launch will apply intense g-forces, it will vibrate and shake at different rates as it goes through different stages and the satellite is ejected on the final stage. To withstand these different stages a satellite and its component parts undergo extensive testing during the various phases of its development.

Some companies in the space industry are using low-cost satellites to perform the necessary tests and collect information on the parameters that characterize the environment in which their new technologies will operate.

For CubeSats, NASA has one document titled "Program Level Dispenser and CubeSat Requirements Document" that gives a set of NASA specifications on the requirements for CubeSat and PPOD. They include vibration, shock, and thermal requirements as well as power limits during phases of launch.

NASA has several facilities to perform spacecraft tests.

The majority of European Space Agency (ESA) spacecraft are tested in the European Space Research and Technology Centre (ESTEC). This test centre is the largest centre of its kind in Europe, and one of the largest in the world.

2.4.1 Clean Rooms

Clean rooms are like operating rooms in hospitals, they are usually kept as free as possible of contaminants that could interfere with delicate technology. That way is maintained by having filtered air pumped in and maintaining positive pressure so no other outside air can enter. Some clean rooms can fit a satellite and the team needed to build it.

Anyone that works in a clean room, such as engineers and scientists, have to wear sterile bodysuits, head covers, gloves, boots, and face masks. Before entering the clean rooms theres is a space where employees receive a forced air shower to blow away loose debris from hair, skin and clothing.

It takes about 10 to 15 minutes to get dressed before entering the clean room, and it's important because even a little of dust or a fingerprint could severely damage the sensitive components and instruments, so the clean room must filter out these harmful contaminants.

In Goddard Space Flight Center there is a clean room with a Class-10,000 according with US FED STD 209E standard. That means any cubic foot of air in the clean room has no more than 10,000 particles floating around in it larger than 0.5 microns.

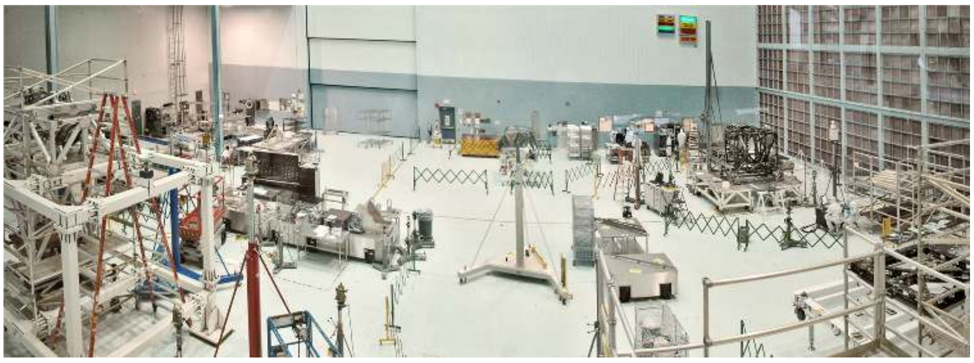


Figure 2.13: Clean room at the NASA's Goddard Space Flight Center [25].

2.4.2 Thermal Vacuum Testing

Thermal vacuum testing is crucially important for spacecraft because it represents the closest possible replication of space conditions capable of being reproduced on the terrestrial surface.

There is no conduction or convection in space, because there is there is absence of air and no material to conduct heat. Only radiative heating and cooling occurs in space. In orbit the satellite will heat rapidly, or cool rapidly, without having any buffer between the two states.

Vacuum testing also ensures that soldered links are stable, and aren't going to break due to air pockets, air conducting where there should be solder, and similar problems. If there are air bubbles in a component due to poor manufacturing, will cause it to break when it hits vacuum [14].

The Johnson Space Center have a thermal vacuum chamber (Figure 2.14) capable of reaching temperatures between $-260°C$ and $150°C$. This chamber has a pressure range of 1×10^{-6} to 760 Torr.

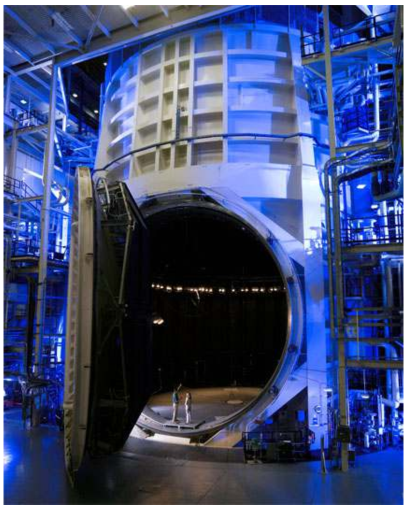


Figure 2.14: Thermal vacuum chamber A at the Johnson Space Center[20].

The Phenix thermal vacuum chamber (Figure 2.15) is inside the ESTEC's facilities. It have a inner box called the the thermal tent that is made of copper plates with brazed copper pipes which use liquid and gaseous nitrogen to produce the range of temperatures desired, from $100°C$ to less than $-170°C$.

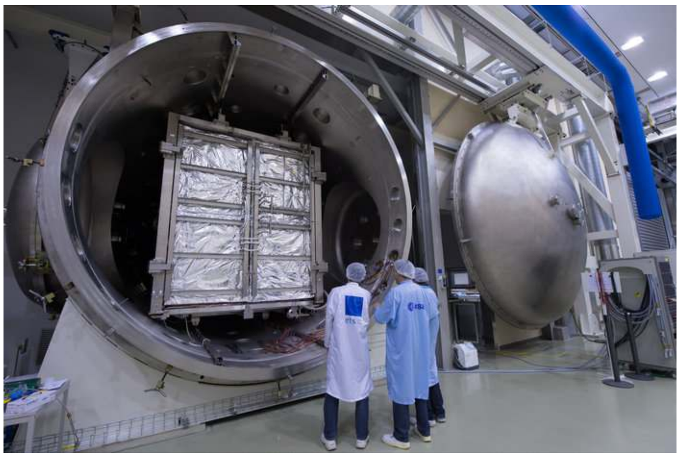


Figure 2.15: Phenix thermal vacuum chamber with thermal tent from ESTEC Test Centre [47].

2.4.3 Vibration Testing

A rocket vibrates with high level of intensity during launching moment, due to engine throttling and wind resistance. Each stage of the rocket will induce changing vibrations from stage shutdowns and stage separations.

Each rocket has its own vibration profile of the range of frequencies and amplitudes that a payload on that rocket will suffer.

The space industry demands the most rigorous vibration testing in the world.The first two minutes of a satellite's space flight are the hardest, as it experiences the extreme vibration at launch. Therefore is essential to test a satellite under similar conditions to ensure that satellite will survive this phase until reaching the orbit.

In NASA's Goddard Space Flight Center, their shaker table (Figure 2.16) can reach vibration with a frequency range of 5 - 2,000 Hz.

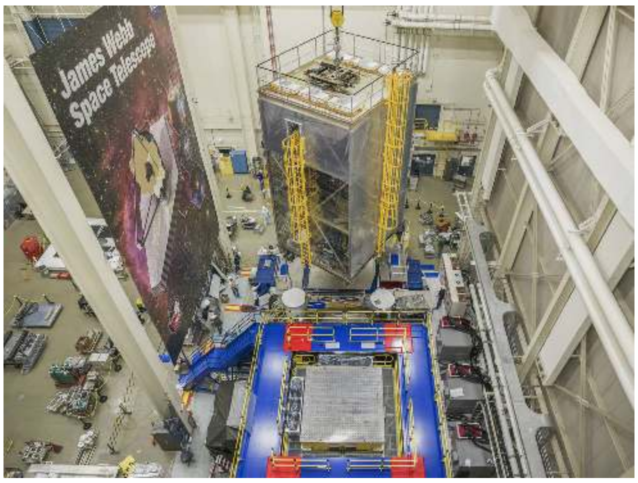


Figure 2.16: James Webb Space Telescope is positioned in the shaker table [46].

The ESTEC Test Centre's Hydra is a hydraulic multi-axis shaker (Figure 2.17) test facility that provides vibration testing. It is capable of generating vibrations equivalent to an earthquake of 7.5 on the Richter scale. Hydra can perform mechanical vibration tests at frequencies between 1 - 100 Hz in all axes of motion simultaneously.

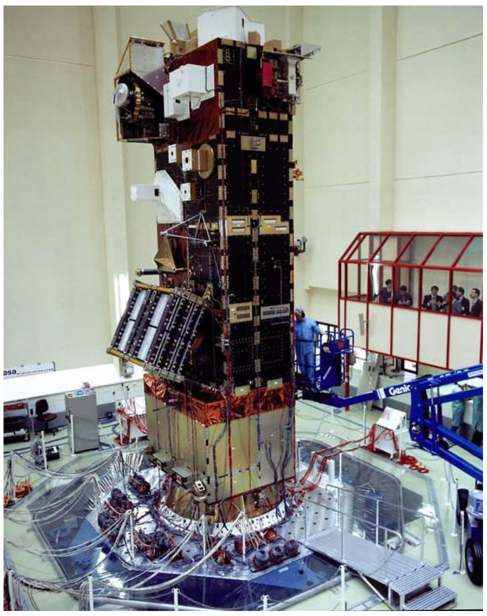


Figure 2.17: Envisat satellite on Hydra [27].

2.4.4 Acoustic Testing

During the launch of a rocket, the release of high velocity engine exhaust gases, the resonant motion of internal engine components, and the aerodynamic flow field associated with high speed vehicle movement through the atmosphere generates elevated amount of noise [14], even when standing several kilometres from the launch pad. A satellite on top of its rocket launcher is exposed to a lot more higher levels of acoustic noise.

That's why engineers have to ensure that the the satellite can withstand that levels of sound pressure.

The Reverberant Acoustic Test Facility (RATF) is located at NASA Glenn Research Center (Figure 2.18). The frequency range is from 25 Hz to 10 kHz and the maximum acoustic overall sound pressure level is 163 dB. The chamber can be operated as a class 100,000 clean room. As similar rooms, while in operation, it is forbidden to be presented inside the chamber, which have thick walls and doors to ensure that there is no sound leakage out of the room.

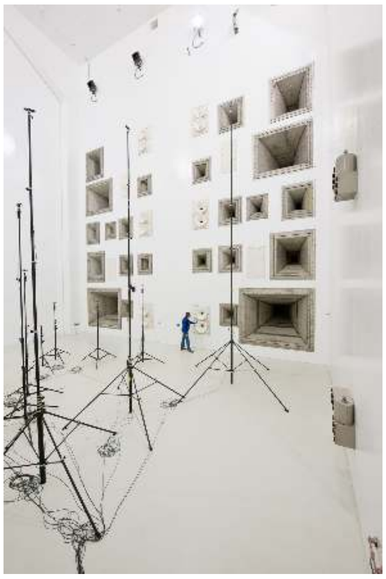


Figure 2.18: Reverberant Acoustic Test Facility [51].

The largest european facility to make acoustic tests is the ESA's Large European Acoustic Facility (LEAF) (Figure 2.19). The noise generation system consists of four different horns with different cut-off frequencies and three high frequency noise generators. It is possible to achieve a maximum overall noise level up to 156 dB.

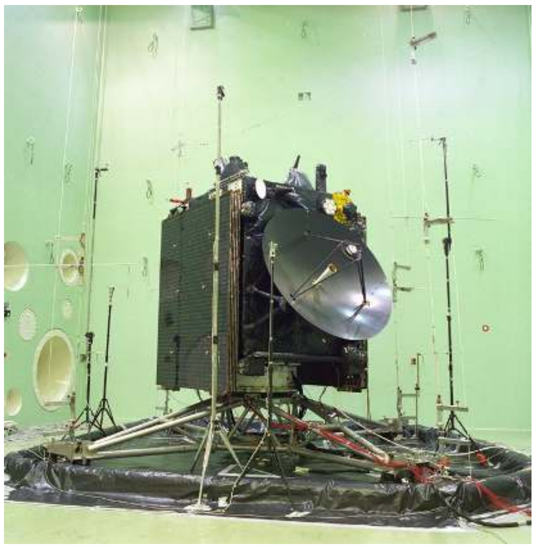


Figure 2.19: ESA's Rosetta satellite during acoustic testing at LEAF [32].

2.4.5 G-Force Testing

With the launch of powerful rockets satellites and astronauts experience huge values of g-force. Satellites can achieve up to 10 G during launch [14].

With this forces it is important to make some tests trying to replicate the stress expect the equipment will go through during launch.

The NASA Ames Research Center have a 20 G centrifuge (Figure 2.20), meaning that is capable of producing forces up to 20 times that of terrestrial gravity, rotating at 50 rpm. This machine was designed in the 1960s. Multiple payloads can be placed along the arms to simultaneously obtain data from hardware at multiple g levels.

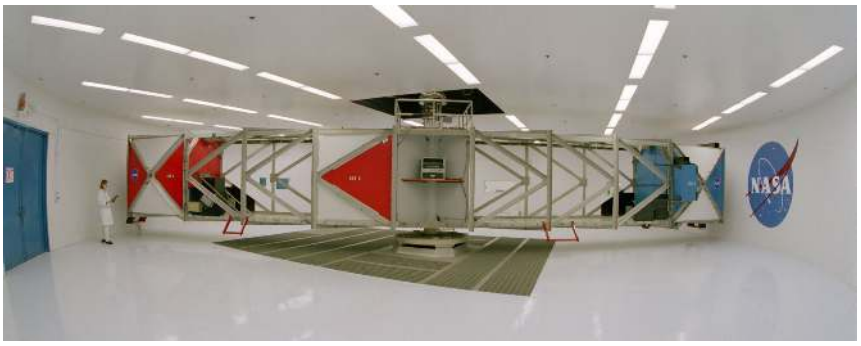


Figure 2.20: Ames Research Center 20 G centrifuge [12].

In the ESTEC facilities, theres is a centrifuge called Large Diameter Centrifuge (Figure 2.21) that can spin at 67 rpm and create gravity up to 20 G.

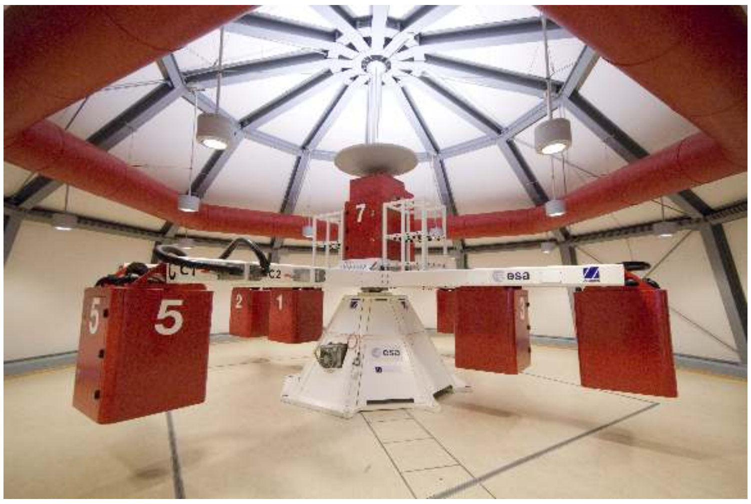


Figure 2.21: Large Diameter Centrifuge at ESTEC [28].

Chapter 3

CTSAT-1 : Design and Implementation

3.1 Structure

As mentioned earlier, a TubeSat is a low-cost satellite. It has a total weight of 750 g, with 250 g being reserved for the additional load associated with the material needed to carry out the experiments.

It's light weight is due to the picosatellite structure, made from a set of printed circuit boards with hexadecagonal shape.

There are in total five levels, four basic levels plus one for the payload (figure 3.1). The satellite has a length of 12.7 cm and outer diameter of 8.94 cm.

In the kit provided by IOS, it includes PCB Gerber files for the four basic levels, microcontroller Arduino Mini, Radiometrix transceiver and amplifier, solar cells, battery, antenna sections and structural hardware like bearings, screws, washers and nuts.

Although IOS provides the some PCB files, these can be changed if necessary.

To understand the complexity of its structure, IOS recommended building a mock-up with blank PCBs, fiberglass and aluminium sheets. However, it was made one using recent 3D printing technologies (figure 3.2). The plastic used in this structure demonstration was PLA (Polylactic Acid), which is cheap, widely available, easy to print and environment friendly. As demonstration purpose, the side panels were not totally built to better show the inside of the picosatellite.

The dimensions of this mock-up are provided in the annex I.

Its important to mention that the rings in this mock-up were made for M3 screws size, and not the screws provided by IOS.

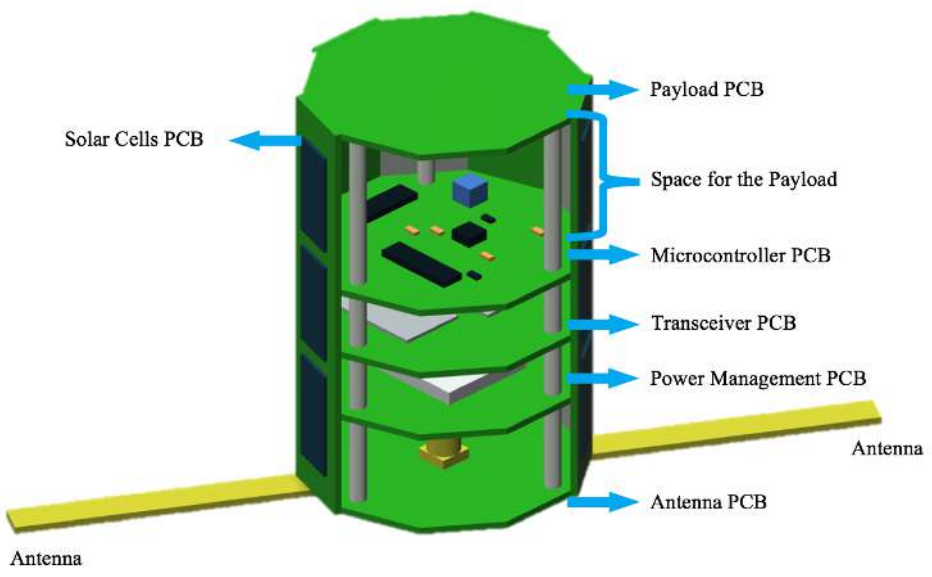


Figure 3.1: TubeSat structure with the five levels and other components.

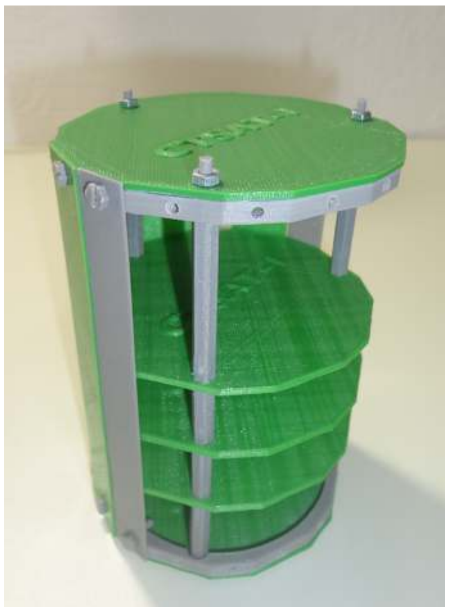


Figure 3.2: TubeSat mock-up using a 3D printer.

The picosatellite uses two rings that secure the solar panels PCBs to the main structure. Although IOS provides indications of materials and dimensions, a reinforced one was made using ABS(Acrylonitrile Butadiene Styrene) in a 3D printer plus a metal mounting system.

As different of IOS procedure, that uses only screws to secure the panels to these rings, this version uses screws and nuts, which these nuts are locked inside the plastic rings.

The screws provided by IOS were in the imperial system, so they were replaced by the closest measure in the metric system to be able to buy the nuts, which was M1.6.

The rings were first made with nuts hole open, then the nuts were placed in the rings and later on those holes were chemically welded with more ABS to secure the nuts.

The dimensions of the rings are provided in the annex I.

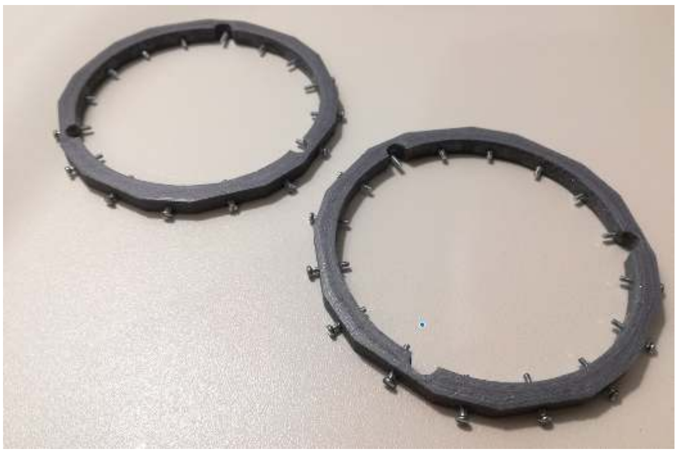


Figure 3.3: TubeSat rings using a 3D printer.

3.1.1 First Level - Antenna

The first level is the Antenna Printed Circuit Board (PCB). Attached to the Antenna PCB are two lengths of antennas made from a steel measuring tape, measuring 33 cm. On one side of the board there is coaxial connector to connect the dipole antenna, and via coaxial cable, connect to the communication level.

Also in the PCB, there is a jumper that must be removed before flight and a switch that activates the satellite after it has been deployed to orbit. The jumper keeps the tubesat turned off, even if the microswitch is activated. Before the picosatellite be put inside a launch cylinder, the jumper is removed as microswitch is not activated because it is pressed against the cylinder walls. When it is launch in orbit the microswitch will activate and turn on the tubesat.

In the Annex II, there is the PCB layout, the circuit schematic and the list of materials.

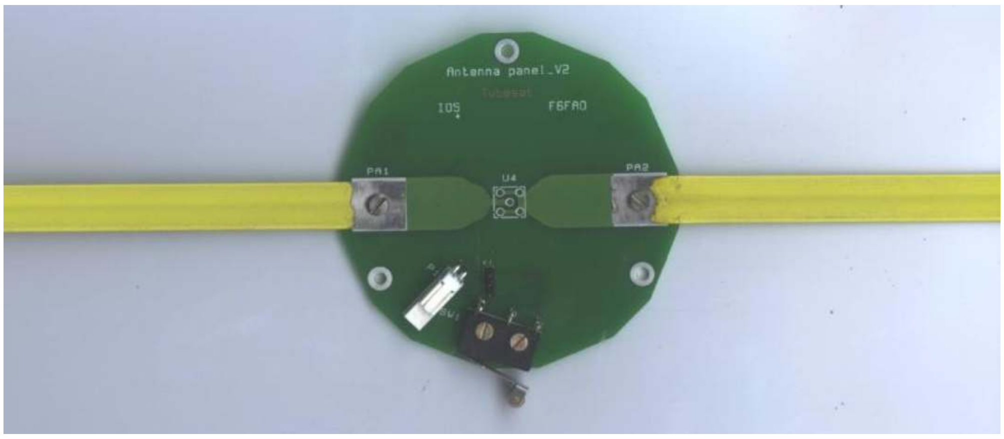


Figure 3.4: First level - Antenna PCB [59].

3.1.2 Second Level - Power Management

The second level is the Power Management PCB. In this board it's included the battery, the solar cell PCB cable connectors, the battery charging circuit, the voltage regulators, and the power connection points for the various satellite systems.

The battery provided is a Li-ion 3.7V with 5200 mAh capacity.

This level is responsible for charging the battery through the energy received by the 48 photovoltaic cells, the battery to power the whole satellite and the management of power connections for other levels.

In the Annex II, there is the PCB layout, the circuit schematic and the list of materials.

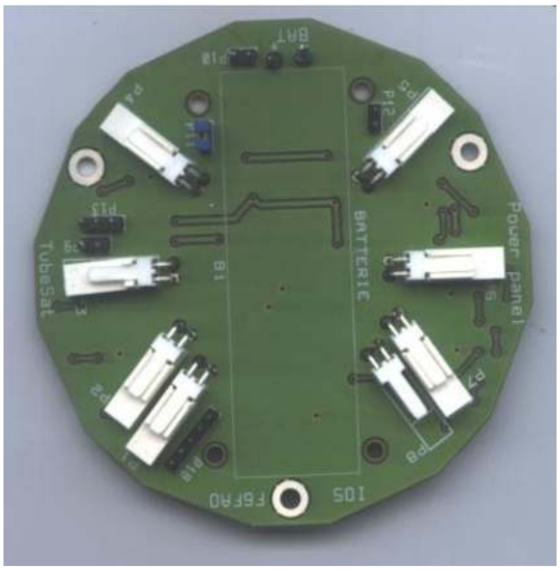


Figure 3.5: Second level - Power Management PCB [59].

3.1.3 Third Level - Communications

The third level is the Transceiver PCB. Here are the transceiver Radiometrix TR2M and the transceiver amplifier Radiometrix AFS2. This board is also connected to the Microcontroller PCB and Antenna PCB.

The output power is about 500mW, and the working frequency is 435 MHz.

The transceiver is a device that is able to transmit and a receive signals in the same package. In the TR2M is possible to reconfigure the entire communication system using the RS232 interface.

The amplifier has a mission to increase the power of the signal sent by the transceiver.

In the Annex II, there is the PCB layout, the circuit schematic and the list of materials.

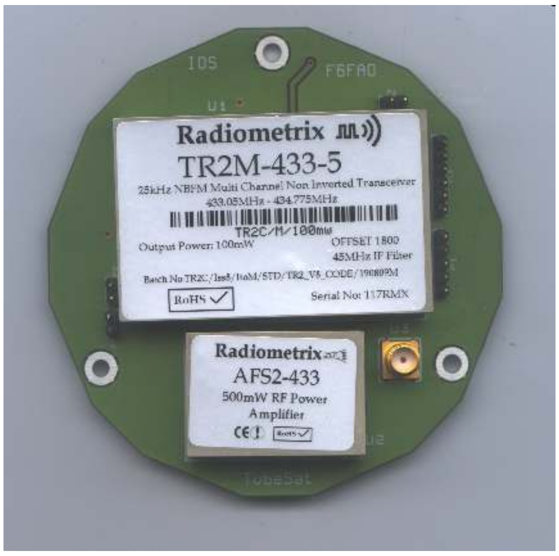


Figure 3.6: Third level - Transceiver PCB [59].

3.1.4 Fourth Level - Control

The fourth level is the Microcontroller PCB. In this board there is the main controller of the TubeSat, the Arduino 5 Mini.

This Arduino have inside a ATMEL microprocessor, which represents a very low energy consumption solution, while maintaining an affordable price.

The objective of this level is process of the collected data and the send them to the communications level.

In the Annex II, there is the PCB layout, the circuit schematic and the list of materials.

3.1.5 Fifth Level - Payload

The space between the Microcontroller PCB and the Payload PCB is reserved for the experiment or application.

Even, if the experiment doesn't require the Payload PCB, it must always be present as a blank PCB, because this board is part of the TubeSat structure.

The payload for the CTSAT-1 satellite is based on a reprogrammable system capable of adapting to the environmental conditions found.

This payload is based on Carlos Loura master thesis titled "Sistema de Aquisição de Sinal para um satélite de baixo custo do tipo Tubesat"[34].

The Payload's block diagram developed for this tubesat is illustrated in figure 3.7.

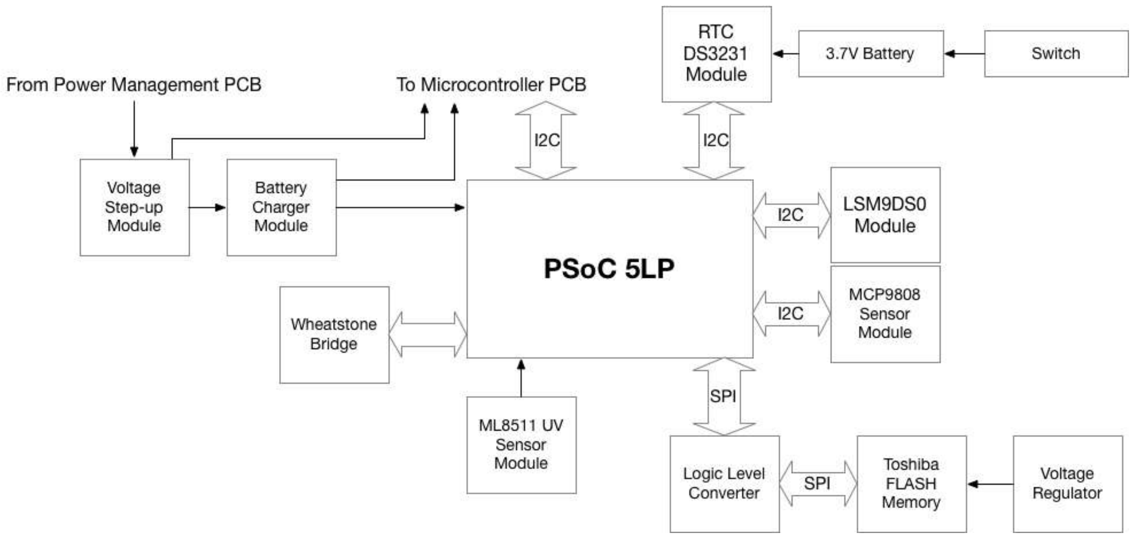


Figure 3.7: Payload's block diagram.

Since one of objectives is to reduce the number of components, it will be used the microprocessor Programmable System-on-Chip (PSoC) CY8C5868AXI-LP035, belonging to the family PSoC5-LP, developed by the company Cypress. This chip has an Advanced RISC Machine (ARM) core, Cortex-M3 CPU, which is a 32-bit processor, with a clock signal frequency up to 67 MHz. This core also has 256 KB of Flash memory, 64 KB of Static Random Access Memory (SRAM), 2 KB of Electrically Erasable Programmable Read-Only Memory (EEPROM). The microprocessor has other features like 20-bit Delta-Sigma analog to digital converter (ADC), 12-bit successive approximation register (SAR) ADC, 4 digital to analog converters (DAC), 62 general purpose input/output (GPIO), etc [49].

As the data keeps reaching from the sensors, it needs to be stored in a memory circuit to send later to Earth station, when the communication channel is available. The memory integrated circuit chosen was TC58CYG0S3HQAIE from Toshiba. This device is based on flash NAND tecnology which is better than the NOR for this application due high-density data storage, low cost-per-bit, write speed and erase speed [43] and these features qualifies better for storage needs. This memory has 1Gb of memory clocked at maximum of 104 MHz. By default, this integrated circuit provide error-correcting code (ECC) logic internally [54]. The communication interface available in this ic is Serial Peripheral Interface (SPI).

As this ic request a power supply from 1,7 V to 1,95 V, it was included a voltage regulator MIC5225-1.8 from Micrel. This regulator provides a fixed 1,8 V with up to 150 mA output current and features low dropout voltage, wide input voltage range, low ground current. Also has protections against reverse battery, against reverse leakage, thermal shutdown and current limit [40].

Furthermore, due to this fact of 1,8 V power supply, the memory integrated circuit need a level shifter between his communicating pins and the communicating pins on the PSoC. It is used the MAX3378 from Maxim Integrated to accomplish this voltage translation. This device allows bidirectional level shifting at maximum of 16 Mbps. There are also features like ±15 kV ESD protection, low quiescent current, low supply current in three-state output mode and thermal short-circuit protection [37].

To provide a timebase for the data acquisition, it was added a Real-Time Clock (RTC) module from Adafruit which relies on the integrated circuit DS3231 from Maxim Integrated. This integrated circuit features a real-time clock with an integrated temperature-compensated oscillator (TCXO) and a crystal. The inclusion of a crystal resonator inside the package enhances the long-term accuracy. The accuracy varies from ±2 ppm to ±3.5 ppm, depending on temperature conditions [19]. This communicates with the PSoC through I^2C. The backup battery of the payload is connected to the battery backup input of this device.

The main power supply for the payload PCB comes directly from the 3,7 V Li-Ion battery on the power management PCB. So there is a need to increase this voltage to 5 V to provide power to the payload devices. In this case is used the module POWERBOOST 1000 BASIC from Adafruit which includes the boost converter chip TPS61030 from Texas Instruments. This integrated circuit can provide up to 96% efficiency and up to a maximum of 4,5 A while having a low quiescent current [56].

To charge the payload battery, is used the module POWERBOOST 1000C from Adafruit. This module is based on the integrated circuits TPS61090 from Texas Instruments and MCP73871 from Microchip Technology. The chip TPS61090 is a boost converter that can achieve delivery current up to 2,5 A, features low quiescent current and high efficiency [57]. The chip MCP73871 is a system load sharing and Li-Ion / Li-Polymer battery charge management controller. The load sharing feature means that the chip can simultaneously power a load and charge the Li-Ion battery. There also other features as voltage proportional current control (VPCC) to ensure the load has priority over Li-Ion battery charge current, constant current / constant voltage (CC/CV) charging algorithm, high accuracy charge voltage, etc [38].

During the picosatellite launch into space, all electronics must be turned off and that is ensured by a deployment switch present in the power management board. However, the backup battery on the payload is not turned off, so its necessary a switch to ensure a properly disconnection from the rest of the payload devices. That is ensured by the integrated circuit STMPS2171 from STMicroelectronics. This switch features 1000 mA maximum continuous current, thermal, short-circuit and reverse current protections as well as low R_{on} [53].

The backup battery used in this payload is the GSP 532248 Li-Polymer 3,7 V 500 mAh from Nimo Electronic with dimensions 53 x 22 x 4,8 mm.

As one of the objectives is to collect some data from space, namely temperature, uv radiation, magnetic fields information, various sensors were chosen to accomplish that tasks.

For ultraviolet (UV) light radiation measurements was chosen a module from SparkFun Electronics which contains the ML8511 UV sensor from Lapis Semiconductor. This sensor is equipped with an internal amplifier , which can convert the photo-current to voltage depending on UV intensity, to be able to interface with an ADC. Also provides sensitivity to UV-A and UV-B radiations with low supply current and low standby current. It can detect wavelength from 280 to 390 nm most effectively [42]. The characteristic between output voltage and UV intensity can be found on figure 3.8. This sensor is powered by a 3,3 V power supply, which is taken from other sensor that have a regulator with this voltage.

The UV radiation is part of the electromagnetic spectrum that reaches the Earth from the sun. This radiation is invisible to naked eye, due the lower wavelength than visible light. Most of this UV radiation that penetrates the atmosphere, divides into UV-A (tanning rays) with a wavelength of 320 - 400 nm and UV-B (burning rays) with a wavelength of 280 - 320 nm [16].

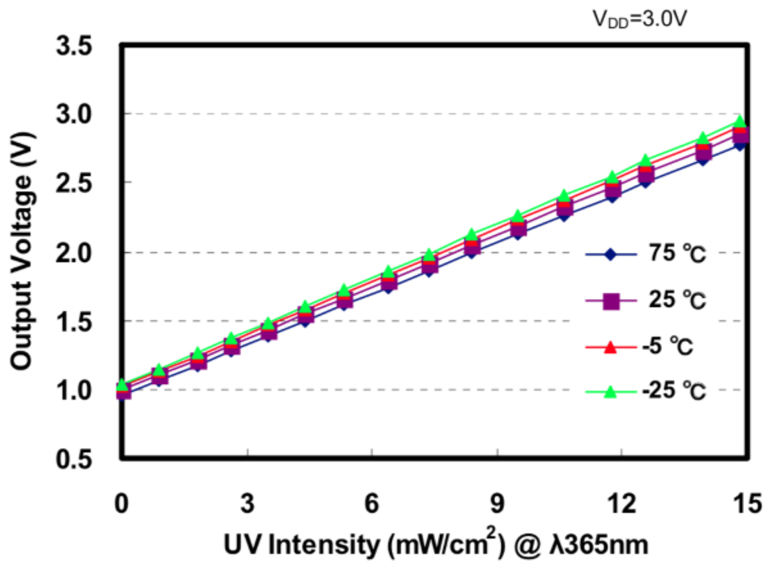


Figure 3.8: Output voltage - UV intensity characteristics of ML8511 sensor [42].

To capture the magnetic field information is used a module from Adafruit that have the integrated circuit LSM9DS0 from STMicroelectronics. This chip has included a 3D accelerometer sensor, a 3D magnetometer sensor and a 3D gyroscope sensor, that means there are 3 acceleration channels, 3 magnetic field channels and 3 angular rate channels. The linear acceleration, the magnetic field and the angular rate have various selectable scales. There are programmable interrupt generators and the sensors can be set in power-down mode separately to save power [35]. The communication between the PSoC and this ic is done through I^2C serial bus interface. As this module has the MIC5225-3.3 regulator to provide power to the integrated circuit also has an output pin from this regulator, which in this payload will power the UV sensor.

The temperature measurements relies on two different methods, one is through a digital temperature sensor integrated circuit the other is an analog temperature sensor. The objective of having an analog sensor is to evaluate its precision versus a digital integrated circuit.

For the digital temperature sensor ic, is used a module from Adafruit that integrates the MCP9808 from Microchip Technology. The accuracy of this ic is typically $\pm 0,25°C$ from $-40°C$ to $+125°C$. The are various selectable measurement resolutions, with its best resolution as $+0,0625°C$, as well as a programmable temperature alert output [39]. This integrated circuit will communicate with PSoC through I^2C.

For RTD device, it was chosen a platinum SMD flat chip PTS1206M1B1K00P100 from Vishay manufacturer. This component provide short reaction times and great stability of temperature characteristic over time [50]. The datasheet of RTD gives the information for how the data can be handled. Its possible through the provided equations or through the table (table 3.1).

In the equations method, temperatures between -55°C and 0°C is applied the Callendar-Van Dusen equation 3.1 and for temperatures between 0°C and +175°C the equation 3.1 can be simplified to equation 3.2 [55]. At 0°C, the value is the nominal resistance of the RTD at this temperature which is 1000 Ω.

$$R_T = R_0 \times (1 + A \times T + B \times T^2 + C \times (T - 100) \times T^3) \tag{3.1}$$

$$R_T = R_0 \times (1 + A \times T + B \times T^2) \tag{3.2}$$

T represents the temperature in °C. R_T is the resistance as function of temperature and R_0 is the nominal resistance value at 0°C.

A, B and C are coefficients that have following values:

$$A = 3,9083 \times 10^{-3} \quad {}^\circ C^{-1}$$

$$B = -5,775 \times 10^{-7} \quad {}^\circ C^{-2}$$

$$C = -4,183 \times 10^{-12} \quad {}^\circ C^{-4}$$

The linearity of this temperature sensor can be verified on figure 3.9.

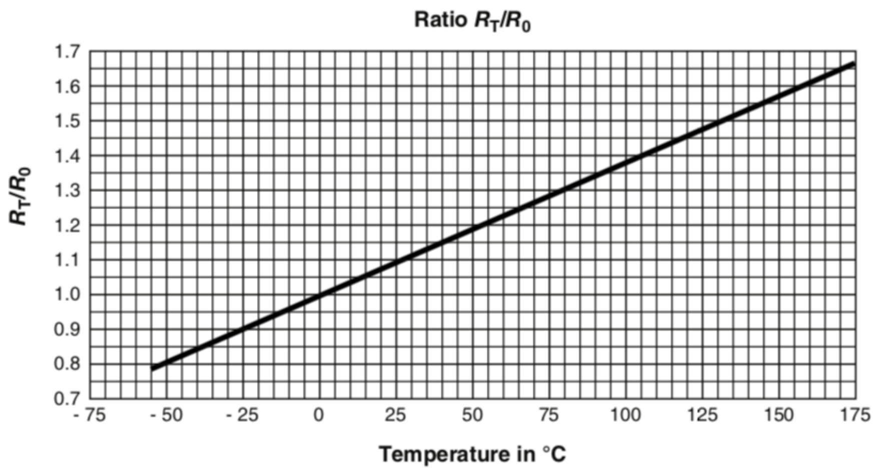
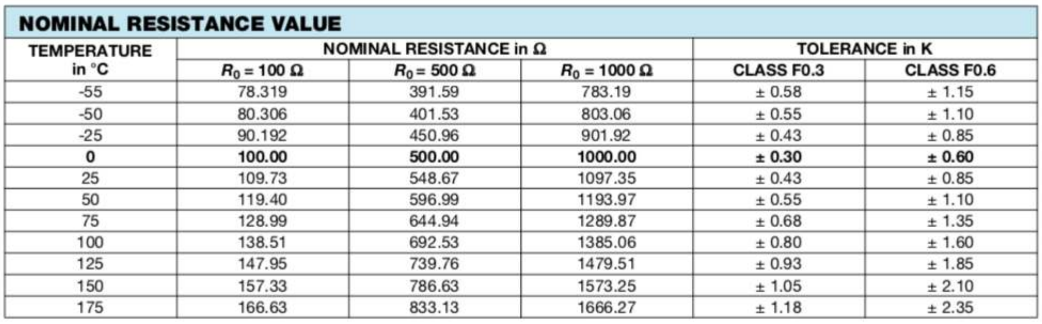


Figure 3.9: Ratio RT/R0 as a function of temperature of RTD from Vishay [50].



NOMINAL RESISTANCE VALUE

TEMPERATURE in °C	NOMINAL RESISTANCE in Ω			TOLERANCE in K	
	R_0 = 100 Ω	R_0 = 500 Ω	R_0 = 1000 Ω	CLASS F0.3	CLASS F0.6
-55	78.319	391.59	783.19	± 0.58	± 1.15
-50	80.306	401.53	803.06	± 0.55	± 1.10
-25	90.192	450.96	901.92	± 0.43	± 0.85
0	**100.00**	**500.00**	**1000.00**	**± 0.30**	**± 0.60**
25	109.73	548.67	1097.35	± 0.43	± 0.85
50	119.40	596.99	1193.97	± 0.55	± 1.10
75	128.99	644.94	1289.87	± 0.68	± 1.35
100	138.51	692.53	1385.06	± 0.80	± 1.60
125	147.95	739.76	1479.51	± 0.93	± 1.85
150	157.33	786.63	1573.25	± 1.05	± 2.10
175	166.63	833.13	1666.27	± 1.18	± 2.35

Table 3.1: Nominal Resistance Value vs Temperature of RTD from Vishay [50].

Instead of putting the RTD directly to an ADC on PSoC, it is used a configuration called Wheatstone bridge (figure 3.10).

To found the value of RTD, it is necessary to resolve the equation 3.3.

$$VRTD - VNRTD = \frac{R1}{R1 + RTD} \times VCC - \frac{R3}{R2 + R3} \times VCC \tag{3.3}$$

Doing some simplification:

$$RTD = \frac{R1 \times VCC}{VRTD - VNRTD + \frac{R3}{R2+R3} \times VCC} - R1 \tag{3.4}$$

The values for R1, R2 and R3 where chosen to match the RTD value at 25 °C, that trough the table 3.1 is 1097,35 Ω, which is close to 1,1 kΩ. Prior to soldering, the resistances of resistors were measured. R1 have the value of 1,095 kΩ, R2 have the value of 1,092 kΩ and R3 have the value of 1,094 kΩ.

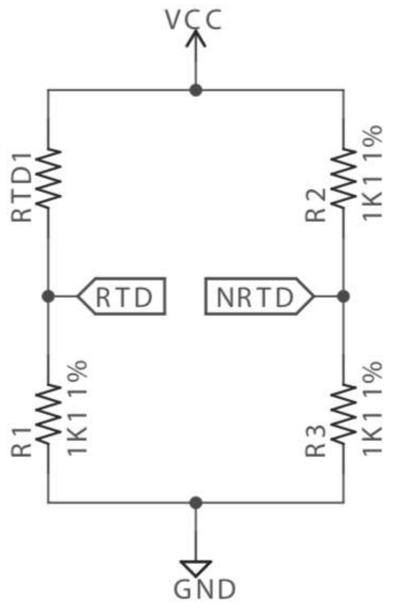


Figure 3.10: Wheatstone bridge used in the payload.

This components were chosen to accomplish wide temperature range and special conditions to fulfil space adverse conditions. Also some packages were chosen had in mind an easy soldering process, which could be done by hand.

To design the board's layout is necessary to choose where each component will be located.

On the top side of the PCB, which is the outer side, is located the sensor modules, such as the UV sensor module, the inertial module, and the digital sensor module. The board also has other modules, such as the RTC module, the voltage step-up module, the battery charger module. The temperature analog sensor is also on this side of the board with the Wheatstone bridge (figure 3.11).

On the bottom side, is located the microprocessor, the flash memory, the logic level converter for the flash memory, the voltage regulator for the flash memory, the switch for the battery, the battery, and passive components such as resistors and capacitors. There are also some smd jumpers as well as the headers for connecting the battery, the I^2C communication header, the main power supply header for the board, the power control header which allows the microcontroller PCB to shutdown the payload and the programming/debugging header (figure 3.12).

The decoupling capacitors on this board, were placed as close as possible to the microprocessor and the flash memory voltage regulator.

As there are no huge power supply requirements for the modules, most of the traces on the board are 0,25 mm width. However there are also 0,4 mm and 0,6 mm width traces for the power supply connections. Around the microprocessor it was created a copper pour to make a ground plane.

In the Annex II, there is the PCB layout, the complete circuit schematic and the list of materials.

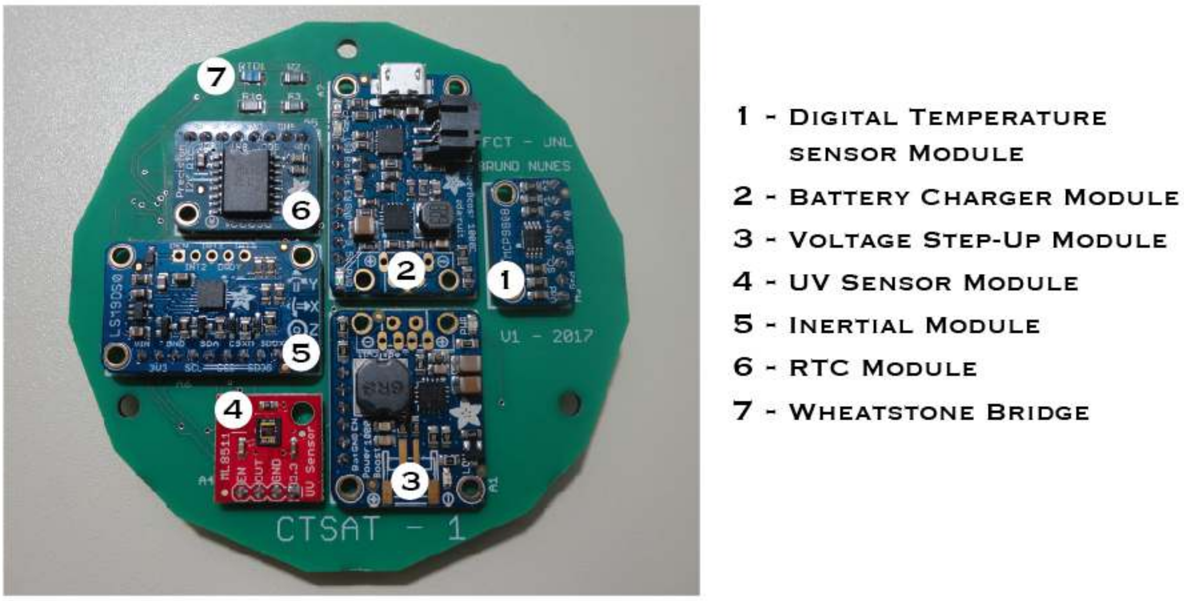


Figure 3.11: Fifth level - Payload PCB - Top view.

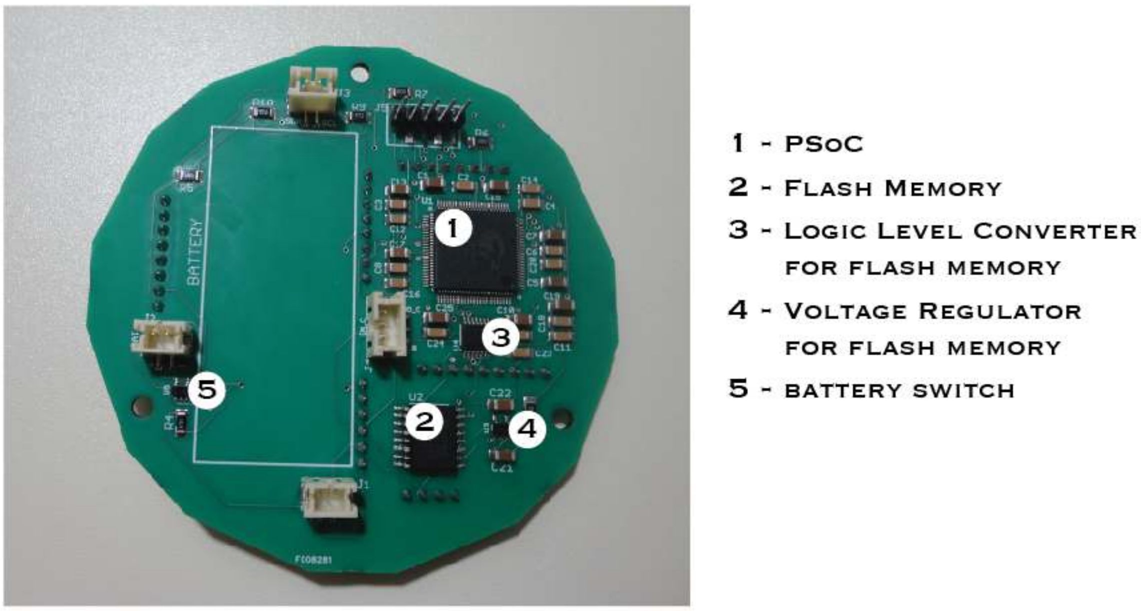

1 - PSoC
2 - Flash Memory
3 - Logic Level Converter for flash memory
4 - Voltage Regulator for flash memory
5 - battery switch
BATTERY

Figure 3.12: Fifth level - Payload PCB - Bottom view without battery.

3.1.6 Solar Cells PCB

As other satellites of its kind, the power source to maintain autonomy is the photovoltaic cells.

TubeSat has a total of 48 cells divided into 8 PCBs. Each solar cell is a triple junction Triangular Advanced Solar Cell (TASC) made by Spectrolab with 2.52 V and 31 mA and its efficiency is around 27 %.

All the photovoltaic cells are connected in parallel and connected to the power management level, which will then distribute to the other levels of the picosatellite.

In the Annex II, there is the PCB layout and the circuit schematic.

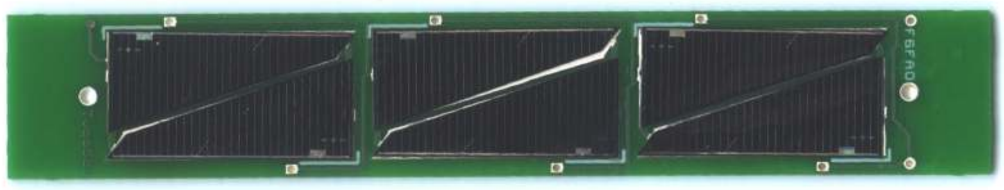


Figure 3.13: Solar Cells PCB [59].

3.2 System Implementation

3.2.1 Startup

After the picosatellite's launch into space, the device turns on with the activation of a mechanical switch. From that time the Arduino's system and PSoC's system begin to start the first proceedings.

The startup of PSoC relies on figure 3.14, that represents the method for that startup.

The RTC synchronisation is very important for data, but not mandatory if the system can't acquired a proper time definition. After some time the system moves to data reading with periodical tries to sync time with main controller and Earth station.

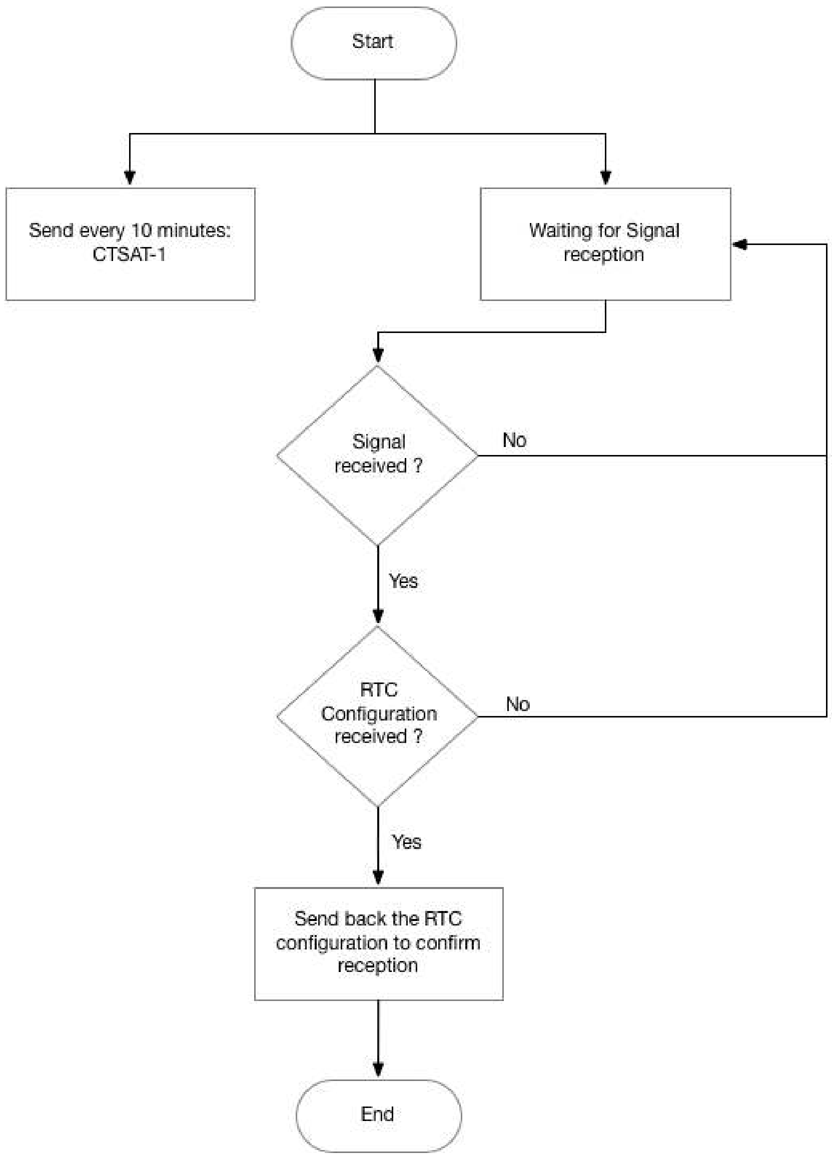
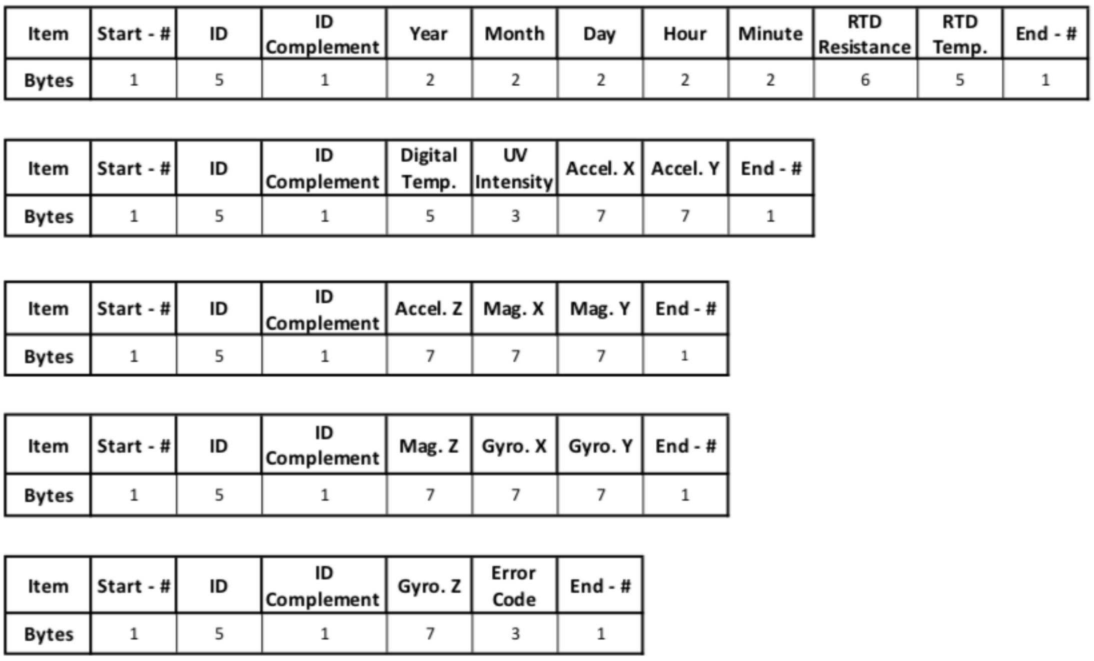

Start
Send every 10 minutes: CTSAT-1
Waiting for Signal reception
Signal received ?
No
Yes
RTC Configuration received ?
No
Yes
Send back the RTC configuration to confirm reception
End

Figure 3.14: CTSAT-1 startup process.

3.2.2 Data Format

After the PSoC is fully functional, the data can be acquired from the sensors. This data needs to have a way to structure the information to save later. The data is first acquired and then is applied the data format of figure 3.15. This format is ready for the I^{2}C transmission when available.



Item	Start - #	ID	ID Complement	Year	Month	Day	Hour	Minute	RTD Resistance	RTD Temp.	End - #
Bytes	1	5	1	2	2	2	2	2	6	5	1

Item	Start - #	ID	ID Complement	Digital Temp.	UV Intensity	Accel. X	Accel. Y	End - #
Bytes	1	5	1	5	3	7	7	1

Item	Start - #	ID	ID Complement	Accel. Z	Mag. X	Mag. Y	End - #
Bytes	1	5	1	7	7	7	1

Item	Start - #	ID	ID Complement	Mag. Z	Gyro. X	Gyro. Y	End - #
Bytes	1	5	1	7	7	7	1

Item	Start - #	ID	ID Complement	Gyro. Z	Error Code	End - #
Bytes	1	5	1	7	3	1

Figure 3.15: Data format to register readings.

To understand better this data format, below there is an example.

Example:
ID: 2520
Year: 19
Month: 2
Day: 19
Hour: 16
Minute: 30
RTD Resistance = 1097,31 Ω
RTD Temperature = 26,81 ^oC
Digital Temperature = 26,82 ^oC
UV = 0,39 mW/cm^2
Accelerometer X = 2,45 m/s^2
Accelerometer Y = 1,32 m/s^2
Accelerometer Z = 2,64 m/s^2
Magnetometer X = 1,02 G

Magnetometer Y = 1,38 *G*
Magnetometer Z = 1,15 *G*
Gyroscope X = 35,93 °/*s*
Gyroscope Y = 2,55 °/*s*
Gyroscope Z = 11,13 °/*s*
Error Code = 100 -> No Error

In the data format that reading is the following:
#025201190219163010973102681#
#0252020268203900002450000132#
#025203000026400001020000138#
#025204000011500035930000255#
#0252050001113100#

The flash memory has space for 125 MB, which is sufficient for one reading every 5 minutes. That reading occupies 135 bytes, in one day of readings takes 38,880 KB of memory space and in 5 months of possible that readings will occupy 6,026400 MB. Also in five months is possible to have 44640 readings on the memory.

The number of total readings is enough to the five bytes of the identification block (ID). As the information of each reading is separated by five strings, an ID complement block was created to differentiate them.

3.2.3 Commands

The Arduino, main controller of the picosatellite, needs to send commands to PSoC to accomplish different functions, such as send a request of data, send a time configuration.

Some commands can be summarised in table 3.2.

Commands	
Code	**Description**
320	Send Data
325	Send Data on Specific Date and Time followed by: 325YYMMDDHHMM
340	Configure Clock followed by: 340YYMMDDHHMMSS
375	Send Error Code List

Table 3.2: Some commands codes implemented.

Without these commands it was not possible to send requests to the payload and consequently not receive any data from it.

3.2.4 Communication

The communication between Arduino (main controller) and the PSoC (payload) starts with the startup process through I^2C protocol.

When the PSoC is communicating with the Arduino is doing it in a slave mode, with Arduino being the master.

However when the PSoC starts to reading time and sensors, this controller change its mode of operation to be master in the I^2C protocol.

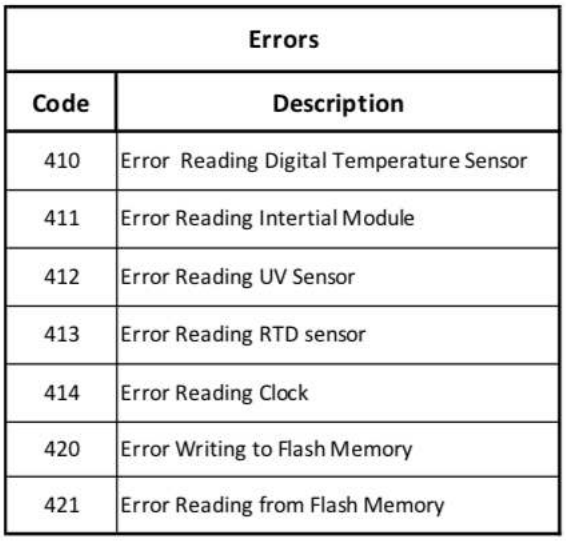
In the payload there is one more communication protocol, that is the SPI. This communication protocol is used to establish a connection with the flash memory.

3.2.5 Error Handling

Although the system was designed to be simple, errors can happen, like every systems.

This errors can appear in the form of sensors and modules malfunction, communications with Arduino fails, memory communications fails, as other fails.

In the table 3.3 there is some error codes.



Errors	
Code	**Description**
410	Error Reading Digital Temperature Sensor
411	Error Reading Intertial Module
412	Error Reading UV Sensor
413	Error Reading RTD sensor
414	Error Reading Clock
420	Error Writing to Flash Memory
421	Error Reading from Flash Memory

Table 3.3: Some error codes implemented.

Chapter 4

Testing Procedures

The testing phase accomplishes the payload components tests with an Arduino and a PSoC as well as other modules tests. Also the complete payload system is tested in this chapter.

To demonstrate a correct operation of a sensor or device, they were first tested with Arduino, because there are already made libraries available. Arduino is one of the most known electronic prototyping platforms. Most of their boards have a programmable microcontroller inside and a built-in programmer to connect directly to Universal Serial Bus (USB) port of computer.

The most important factor of this first tests approach is that almost all of the modules used in this payload have already some libraries ready for Arduino environment.

After testing with the Arduino platform, the testing advanced to a more versatile solution which was the PSoC 5LP CY8CKIT-059. This board provides the same PSoC family as the one that is used in the payload. In this case, it was necessary to create individual libraries for each module under testing.

4.1 Testing Payload Components with Arduino

For this bunch of tests, it was created a testing platform around the Arduino Uno board (figure 4.1). Besides the Arduino UNO, this platform still as a red led, a yellow led, a green led, three input buttons and an alphanumeric LCD 16x2.

For some tests, the main power supply used was a 12 V power supply, for other tests it was used power from USB port.

The complete schematic of this testing platform can be found on annex III.

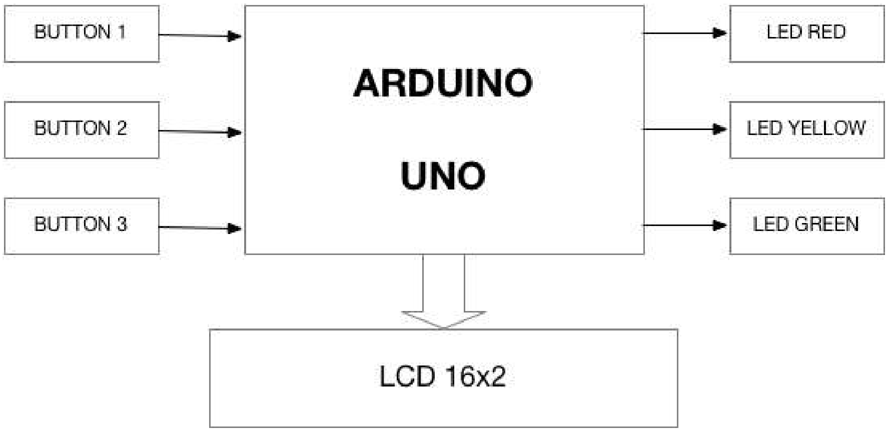


Figure 4.1: Block diagram of testing platform using Arduino.

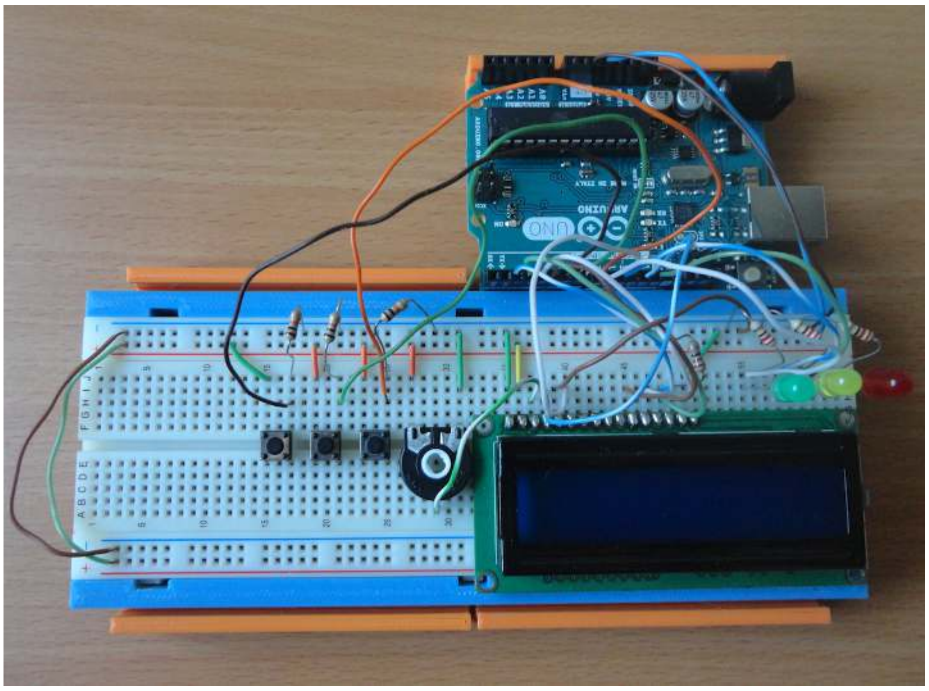


Figure 4.2: Testing platform using Arduino.

4.1.1 RTD Temperature

For the RTD tests built around the Wheatstone bridge, it was chosen two types of experiments due to impact of USB voltage in the system under test.

Also for this test, it was created a small homemade PCB to implement the Wheatstone bridge (RTD plus three resistors).

The first experiment uses the USB connection to provide power for the board and to establish a data transmission between the Arduino and the computer.

The second experiment removes the USB connection as the power comes from 12 V power supply and the data is recorded into a SD Card.

The complete schematics and the source codes of this two experiments can be found on annex III.

4.1.1.1 Experiment 1 - USB

On this experiment, only two wires goes to Arduino testing platform (figure 4.3). The data recording is guaranteed by UART communciation with the computer. The code process diagram is found on figure 4.4.

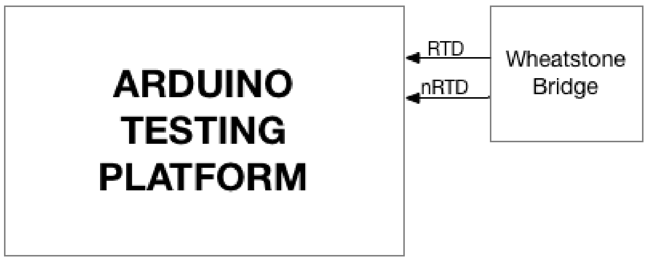


Figure 4.3: Block diagram of RTD experiment 1 test using Arduino.

This experiment follows the procedures presented below.

Procedures:

1. Turn on the power. Run the python script. Leave it for about 3 minutes to sensor stabilise at room temperature.
2. Grab a ice cube in a plastic bag involved in a piece of cloth and put over the sensor. Leave it there for 3 minutes.
3. Remove the piece of cloth and put the ice cube over the sensor for 2 minutes.
4. Spray the sensor with freezing spray for no more than 5 seconds, leave it rest for 5 minutes.
5. Start heating (Thot air = 150ºC low speed) the sensor at a distance of 15 cm and slowly towards the sensor for 4 minutes.
6. Leave it rest for 13 minutes.

After this procedure, the data (RTD values) registered into a file, needs to be converted in temperatures. It is necessary to launch a Matlab script to convert those values. The new values will be stored in a new file.

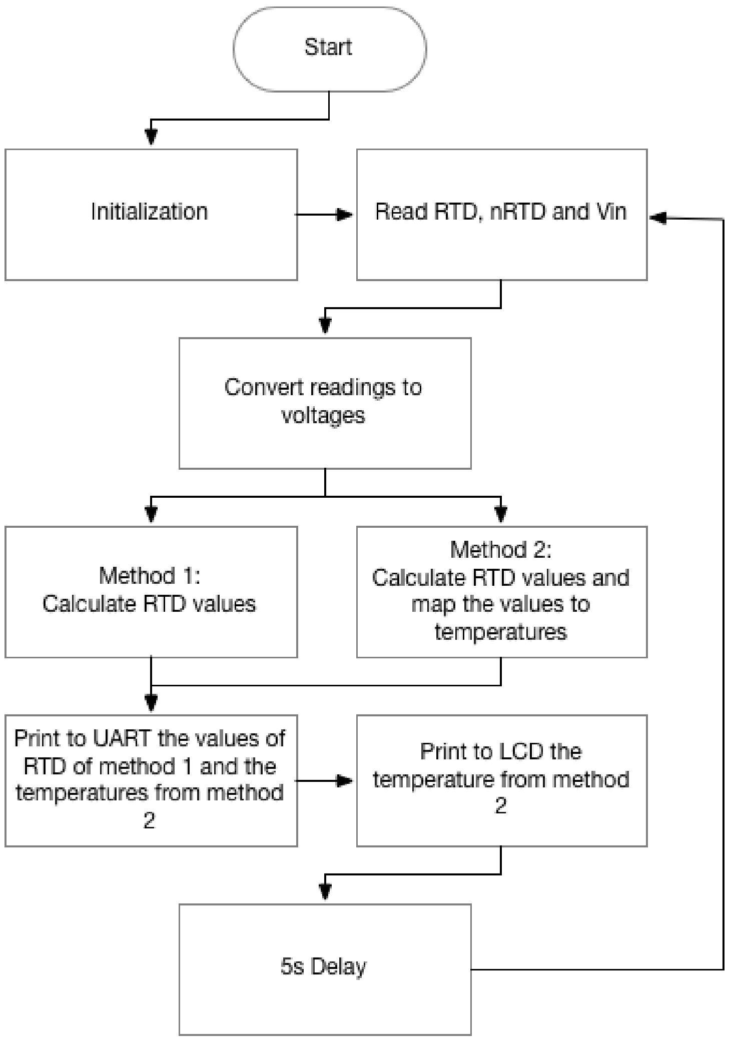

Start
Initialization
Read RTD, nRTD and Vin
Convert readings to voltages
Method 1: Calculate RTD values
Method 2: Calculate RTD values and map the values to temperatures
Print to UART the values of RTD of method 1 and the temperatures from method 2
Print to LCD the temperature from method 2
5s Delay

Figure 4.4: Code process of RTD experiment 1 test using Arduino.

4.1.1.2 Experiment 2 - SD Card

For this experiment, although its similarity with experiment 1, it changes the way how data is handled. As said previously, the data is recorded into a SD Card. However the SD Card requires a 3.3 V power supply, and the Arduino can supply this required voltage but the Arduino's data lines are only 5 V capable. That why it is necessary to add a logic level converter between the SD Card module and the Arduino (figure 4.5) .

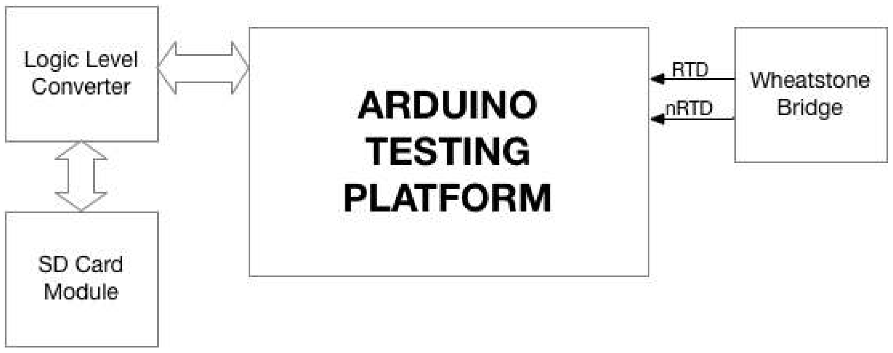


Figure 4.5: Block diagram of RTD experiment 2 test using Arduino.

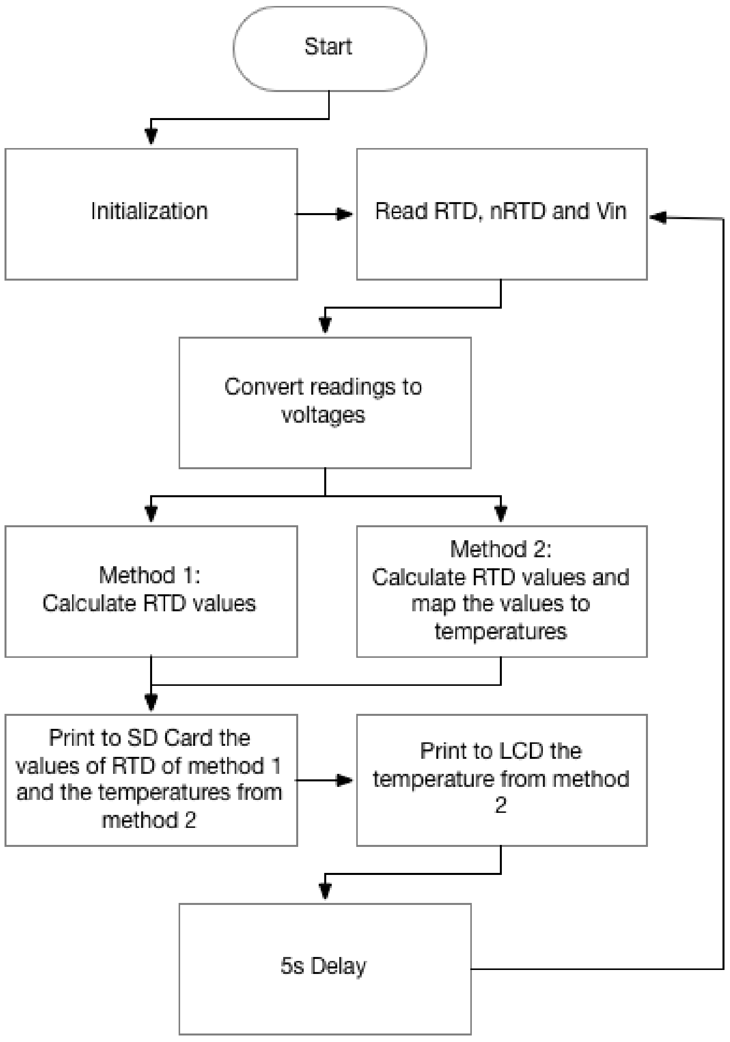

Start
Initialization
Read RTD, nRTD and Vin
Convert readings to voltages
Method 1: Calculate RTD values
Method 2: Calculate RTD values and map the values to temperatures
Print to SD Card the values of RTD of method 1 and the temperatures from method 2
Print to LCD the temperature from method 2
5s Delay

Figure 4.6: Code process of RTD experiment 2 test using Arduino.

The code process is similar to experiment 1 and the procedures are equal.

4.1.2 Digital Temperature Sensor

The digital temperature sensor test contemplate the Arduino testing platform, the sensor and the SD Card record system which is the SD Card module and the logic level converter (figure 4.7). The digital temperature sensor communicates with the Arduino testing platform through I^2C.

The complete schematic and the source code of this test can be found on annex III.

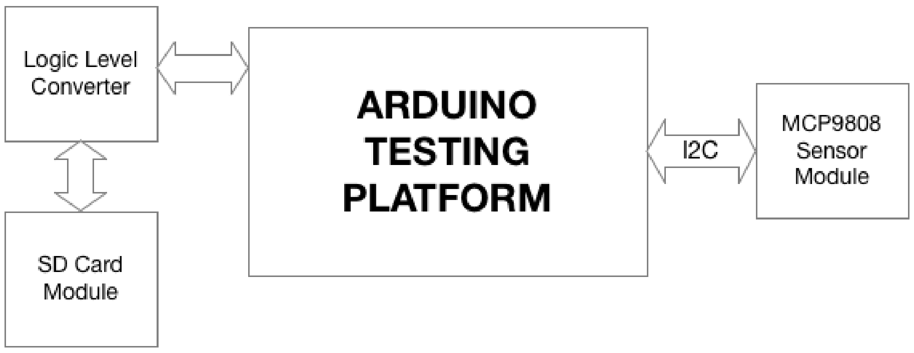


Figure 4.7: Block diagram of digital temperature sensor test using Arduino.

The code process of this test is located in figure 4.8.

The procedures are the same of RTD Tests.

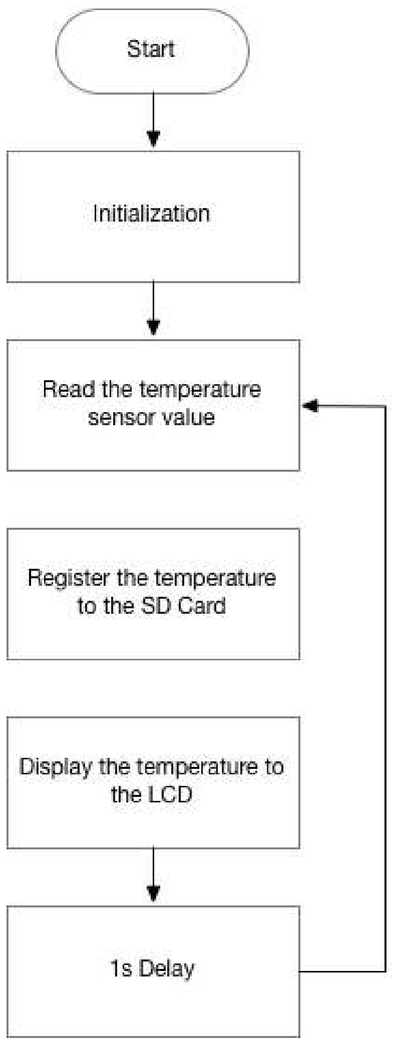

Start
Initialization
Read the temperature sensor value
Register the temperature to the SD Card
Display the temperature to the LCD
1s Delay

Figure 4.8: Code process of digital temperature sensor test using Arduino.

4.1.3 UV Sensor

For this test, the Arduino testing platform receives only the output voltage from the sensor (figure 4.9). As this sensor is powered by Arduino's 3.3 V pin, the maximum voltage that the sensor can output is 3.3 V which is in range of Arduino's analog to digital converter, and that allows for direct connection between the sensor module and the Arduino. The code process diagram is on figure 4.10.

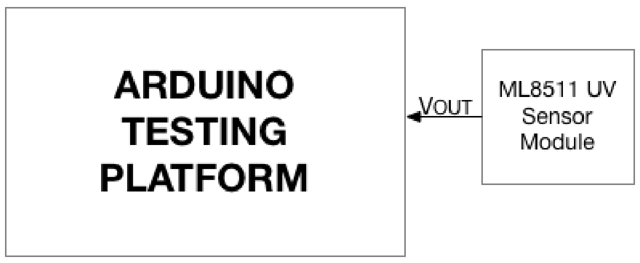


Figure 4.9: Block diagram of UV sensor test using Arduino.

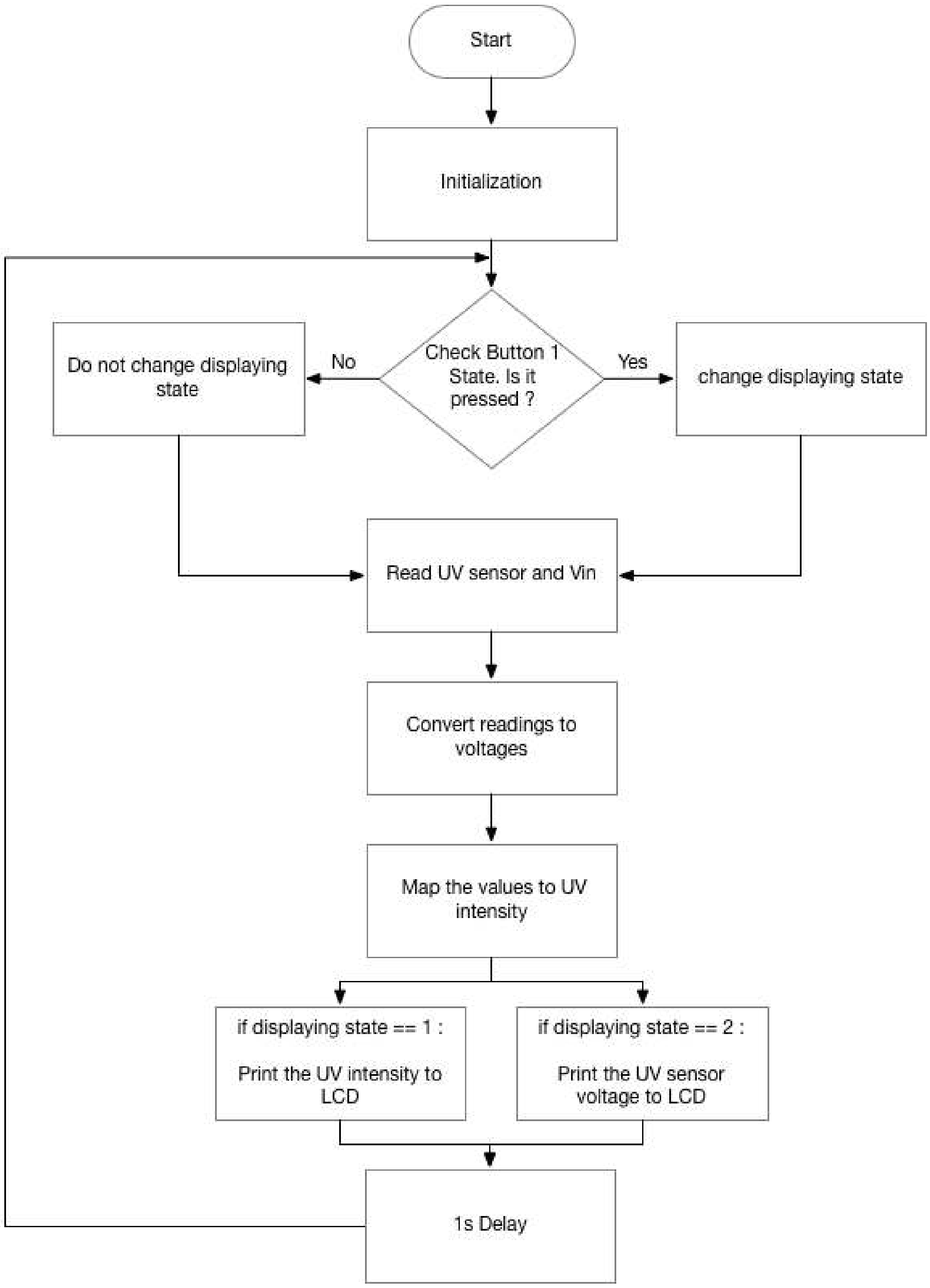

Start
Initialization
Do not change displaying state
No
Check Button 1 State. Is it pressed ?
Yes
change displaying state
Read UV sensor and Vin
Convert readings to voltages
Map the values to UV intensity
if displaying state == 1 :
Print the UV intensity to LCD
if displaying state == 2 :
Print the UV sensor voltage to LCD
1s Delay

Figure 4.10: Code process of UV sensor test using Arduino.

The complete schematic and the source code of this test can be found on annex III.

The procedures for this test are presented below.

Procedures:

1. After power on the system, put the platform or only the sensor in a dark place. Take 3 values from the sensor 30 seconds from each other.
2. Put the system on a normal environment without direct sun light, register 3 values at 30 seconds from each other.
3. Put the sensor in direct sun light. Make 14 measurements at 30 seconds from each other.

4.1.4 Magnetometer, Accelerometer and Gyroscope

This inertial module is connected to the Arduino testing platform though I^{2}C (figure 4.11).

The complete schematic and the source code of this test are located on annex III.

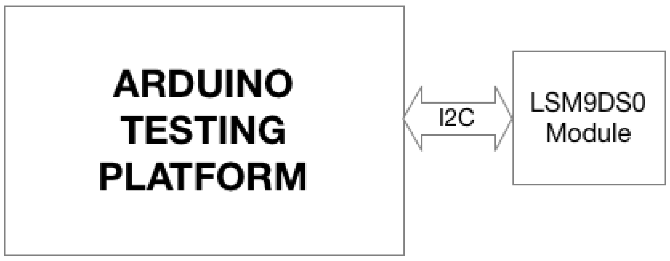


Figure 4.11: Block diagram of magnetometer, accelerometer and gyroscope test using Arduino.

In figure 4.12 is presented the code process diagram for this test.

Below are the procedures for this test.

Procedures:

1. In a stable base (as a table) (figure 4.13 - A), turn on the system wait 30 seconds to sensor stabilise.
2. Accelerometer: Register 5 measurements every 10 seconds.
3. Accelerometer: From the starting point, tilt based on the figure 4.13 - B and register 5 measurements every 10 seconds.
4. Accelerometer: From the starting point, tilt based on the figure 4.13 - C and register 5 measurements every 10 seconds.
5. Accelerometer: From the starting point, tilt based on the figure 4.13 - D and register 5 measurements every 10 seconds.
6. Repeat for magnetometer and gyroscope.

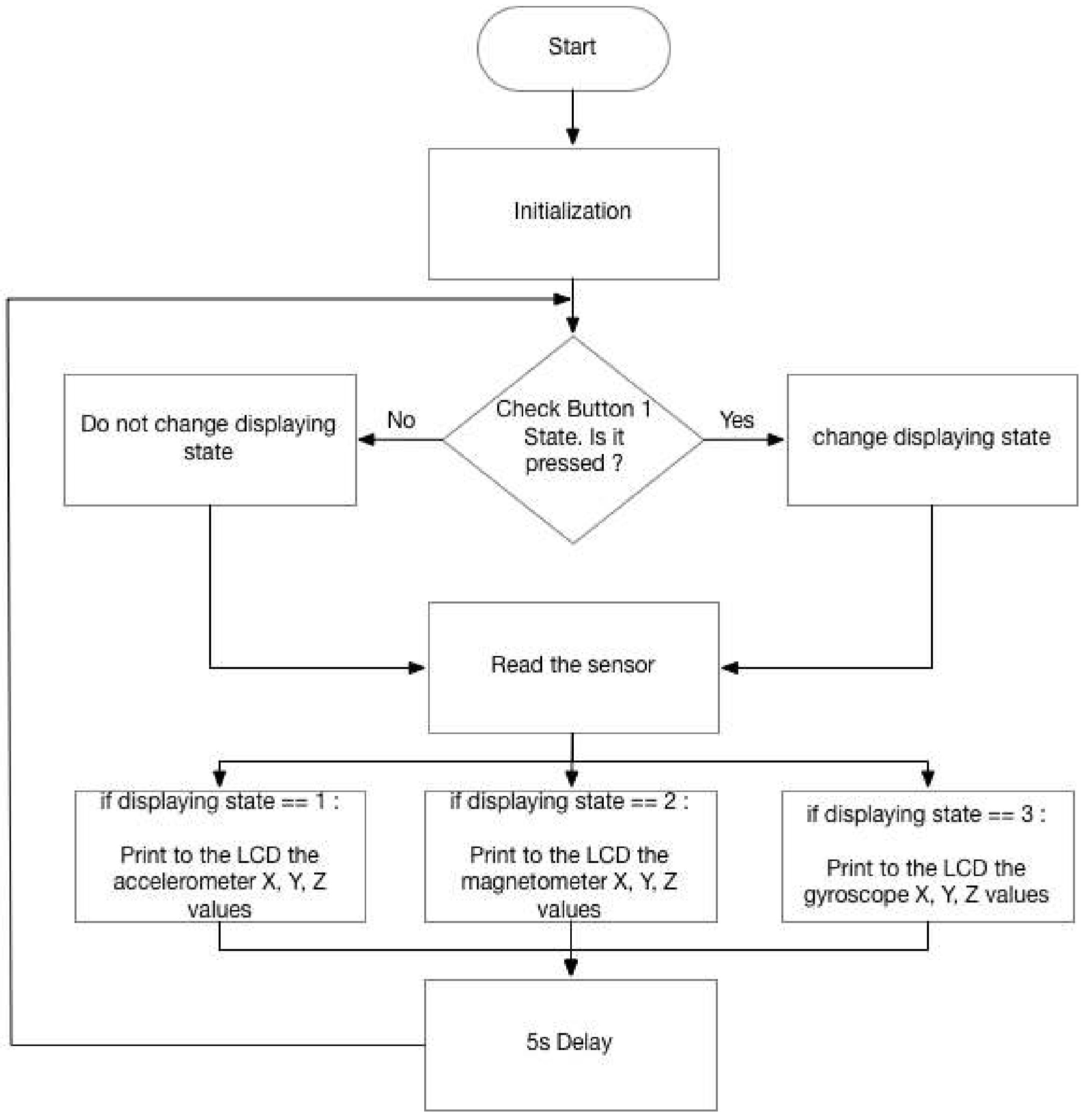

Start
Initialization
Check Button 1 State. Is it pressed ?
No
Yes
Do not change displaying state
change displaying state
Read the sensor
if displaying state == 1 : Print to the LCD the accelerometer X, Y, Z values
if displaying state == 2 : Print to the LCD the magnetometer X, Y, Z values
if displaying state == 3 : Print to the LCD the gyroscope X, Y, Z values
5s Delay

Figure 4.12: Code process of magnetometer, accelerometer and gyroscope test using Arduino.

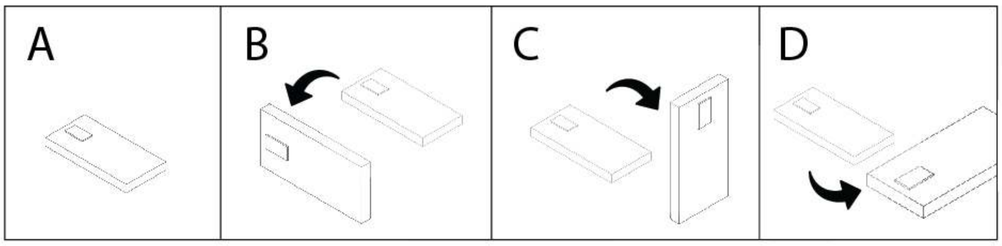


Figure 4.13: Steps for testing the magnetometer, accelerometer and gyroscope. The object represents the testing platform with sensor attached

4.1.5 RTC

An I^{2}C communication is stablished between Arduino testing platform and the RTC. The RTC module requires a 3 V battery for maintaining the clock working even if it loses the main voltage (figure 4.14). It was chosen the CR2032 battery that meets that requirement.

The diagram with the code process is at figure 4.15.

The complete schematic and the source code of this RTC test are located on annex III.

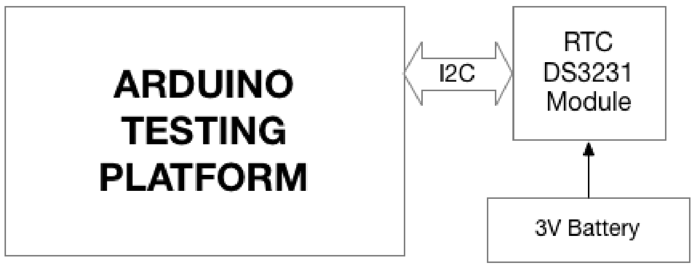


Figure 4.14: Block diagram of RTC test using Arduino.

Below are the procedures for this test.

Procedures:

1. Turn on the system and with a chronometer, register the initial time. After 5 minutes register the time again.
2. Turn off the main power supply and turn on the chronometer for 10 minutes. After the time has passed turn the system back on again and register the time.
3. Compare the values of RTC vs. chronometer.

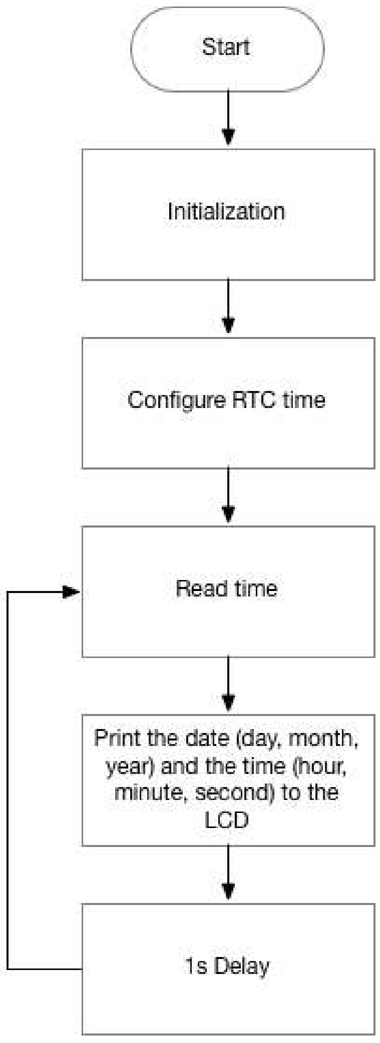

Start
Initialization
Configure RTC time
Read time
Print the date (day, month, year) and the time (hour, minute, second) to the LCD
1s Delay

Figure 4.15: Code process of RTC test using Arduino.

4.2 Testing Payload Components with PSoC

In this tests, it was created a testing platform similar the one that was built for Arduino but with PSoC as the main controller (figure 4.16). This platform has a red led, a yellow led, a green led, three input buttons and an alphanumeric LCD 16x2.

Due this board not having a power supply regulation, the testing platform accommodate a power supply regulation circuit from 12 V to 5 V and 5 V to 3.3 V. This is useful for some tests which were not tested via USB connection.

The complete schematic of this testing platform can be found on annex IV.

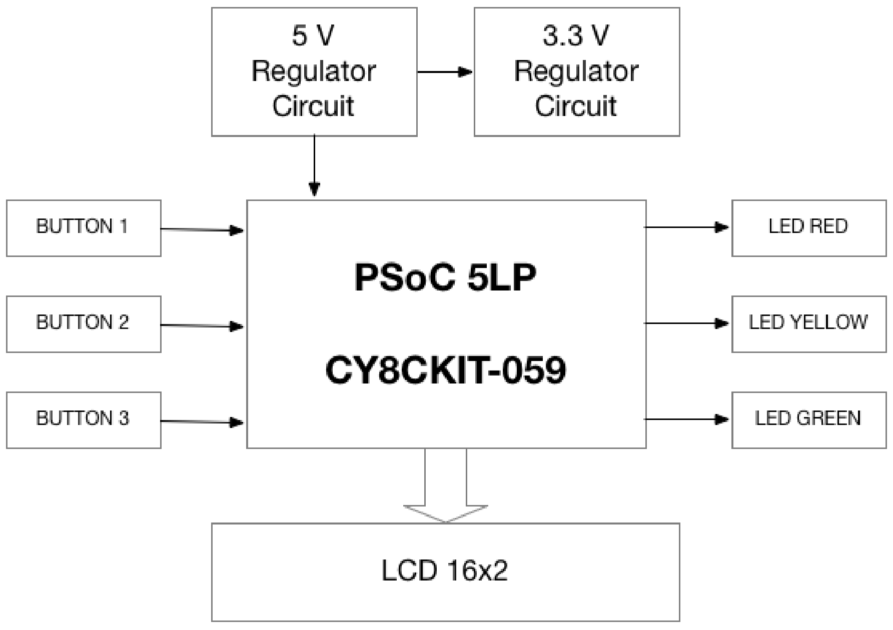


Figure 4.16: Block diagram of testing platform using PSoC.

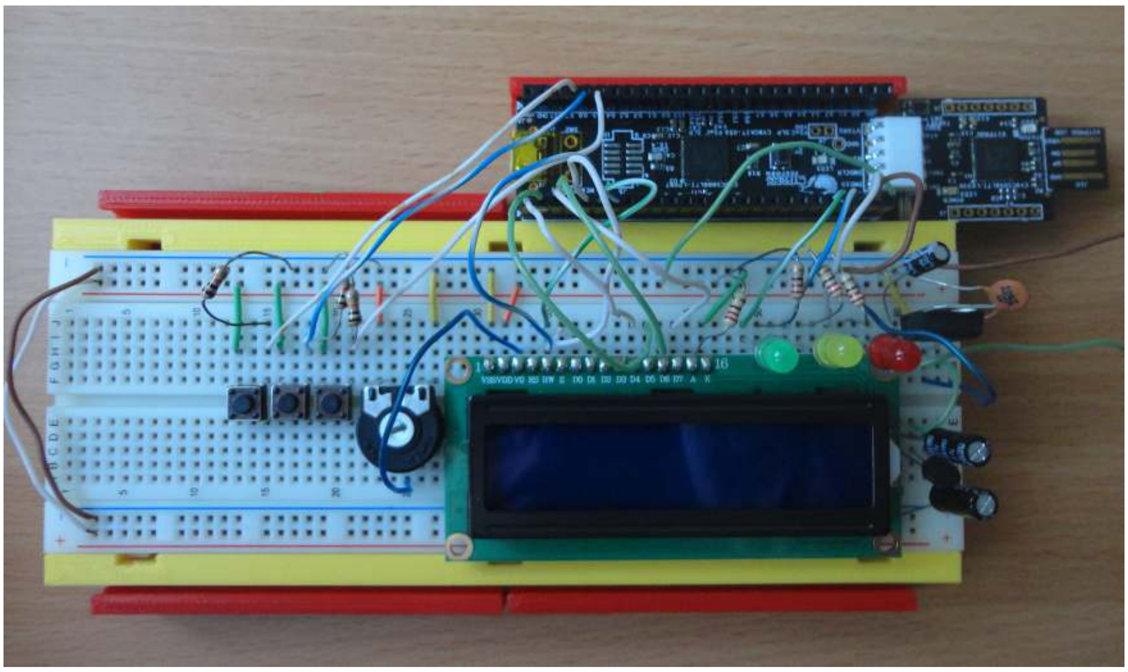


Figure 4.17: Testing platform using PSoC.

4.2.1 RTD Temperature

The RTD temperature sensor test with the PSoC testing platform is very similar to RTD experiment 2 test with an Arduino. However, to use the only Sigma-Delta ADC on PSoC for three input signals, it is necessary to add a multiplexer.

The code process (figure 4.19) is similar to the test with Arduino testing platform.

The complete schematic and the source code can be found on annex IV.

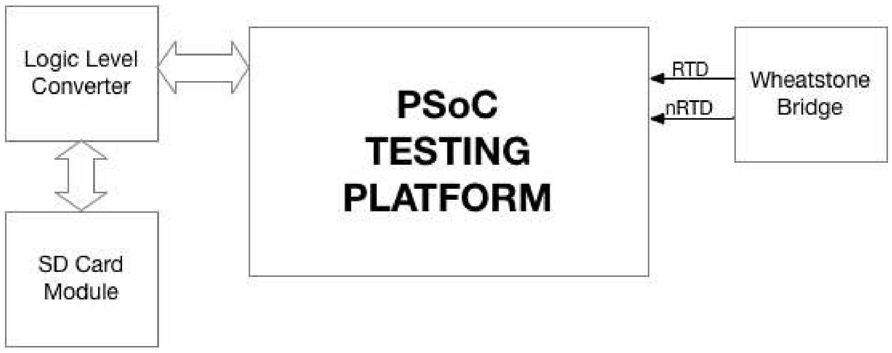


Figure 4.18: Block diagram of RTD test using PSoC.

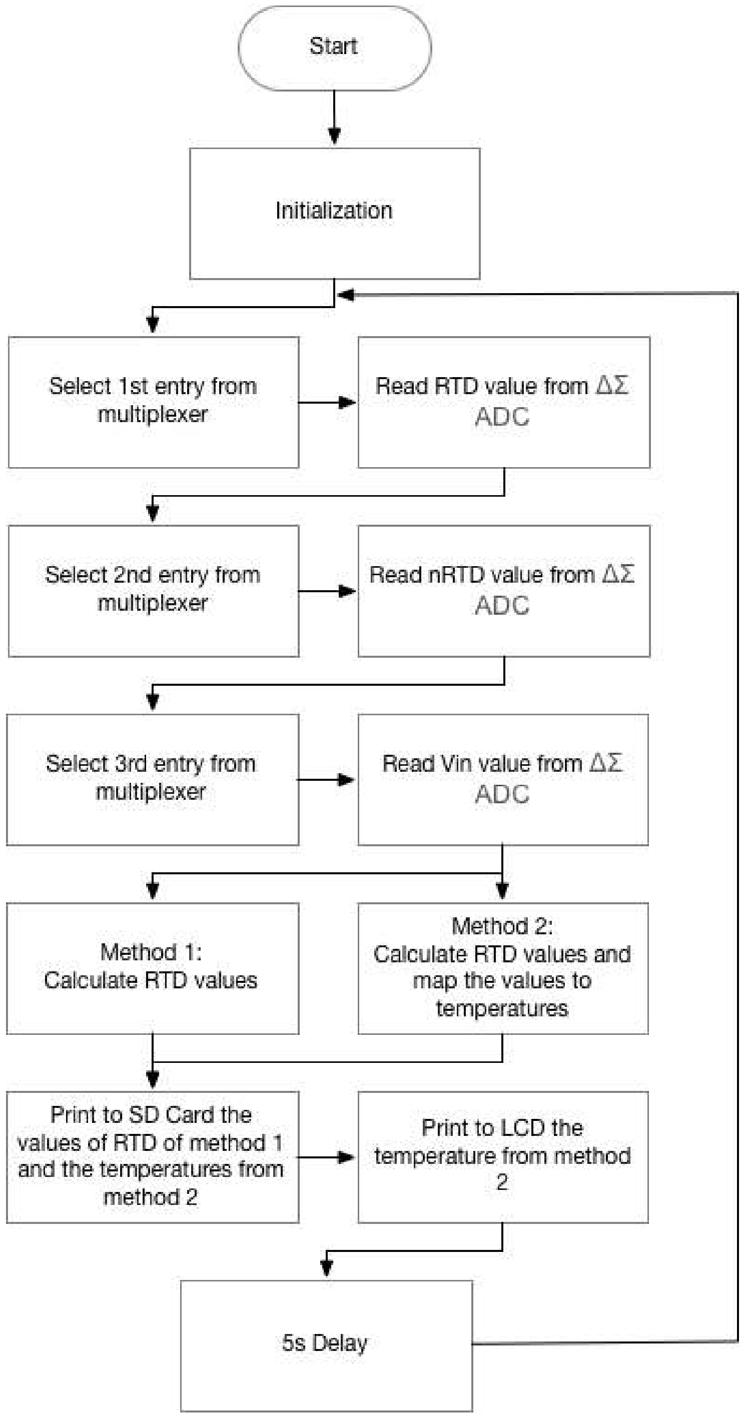

Start
Initialization
Select 1st entry from multiplexer
Read RTD value from ΔΣ ADC
Select 2nd entry from multiplexer
Read nRTD value from ΔΣ ADC
Select 3rd entry from multiplexer
Read Vin value from ΔΣ ADC
Method 1: Calculate RTD values
Method 2: Calculate RTD values and map the values to temperatures
Print to SD Card the values of RTD of method 1 and the temperatures from method 2
Print to LCD the temperature from method 2
5s Delay

Figure 4.19: Code process of RTD test using PSoC.

In PSoC Creator it was created the following blocks that are necessary to this test.

The procedures are equal to procedures of RTD experiment 2 test with the Arduino testing platform.

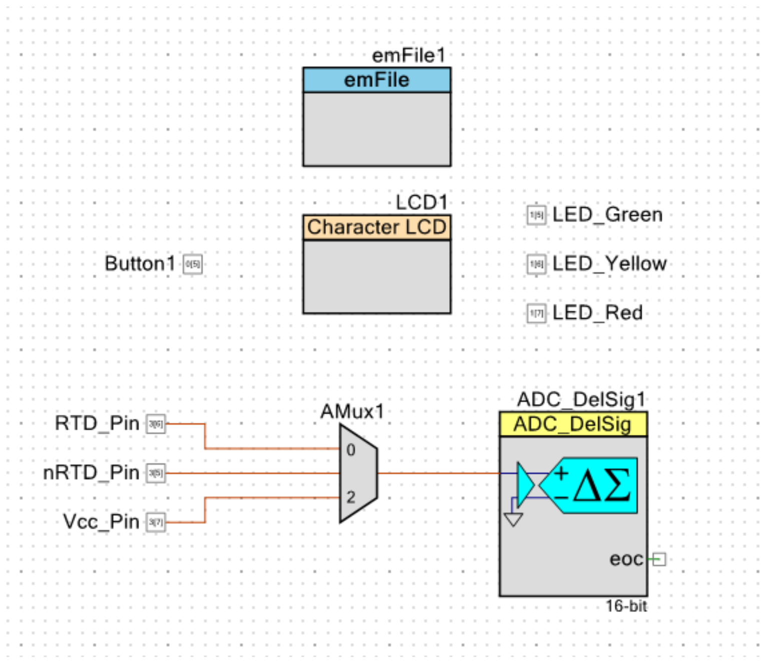


Figure 4.20: PSoC Creator blocks for RTD tests.

4.2.2 Digital Temperature Sensor

This test consists on the same implementation that was done with Arduino, changing only the controllers. The procedures and the code process are equal to the test with the Arduino testing platform.

The complete schematic the source code can be located in annex IV.

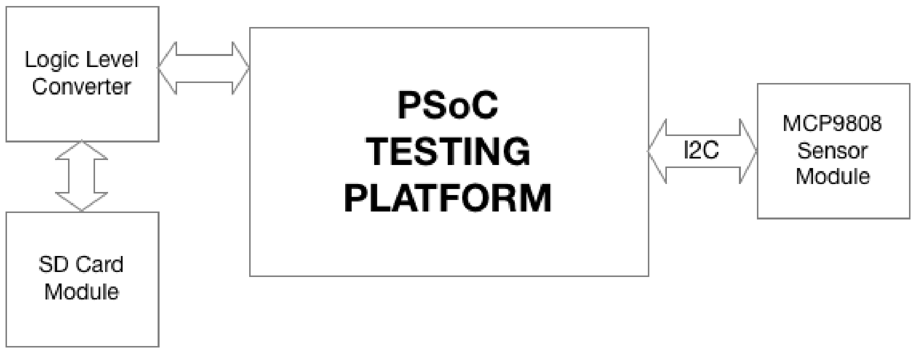


Figure 4.21: Block diagram of digital temperature sensor test using PSoC.

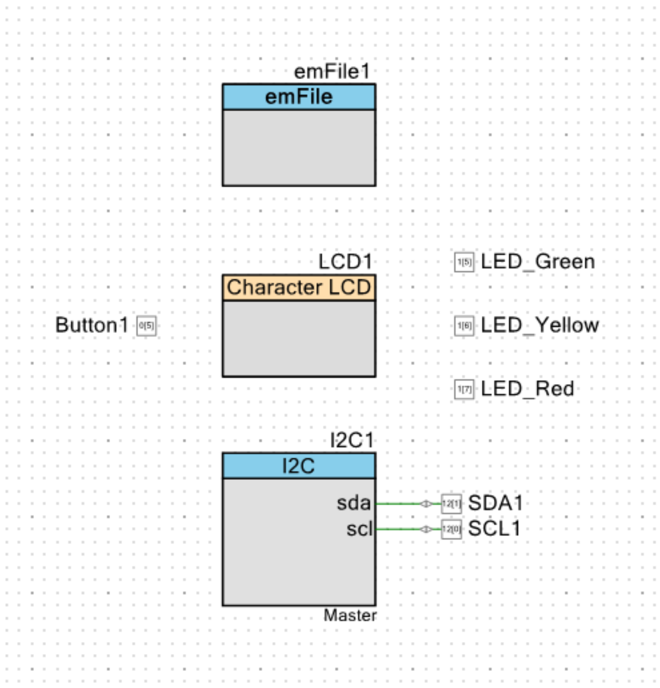


Figure 4.22: PSoC Creator blocks for digital temperature tests.

4.2.3 UV Sensor

The test with the UV sensor is based on the same test with the Arduino testing platform. The procedures and the code process are similar. Due to only one input to the ADC there is no need to add a multiplexer. The Vin input is not used here because the component Delta-Sigma ADC of PSoC creator has already a function to convert automatically the data read to Volt.

In annex IV, there are the schematic and the source code of this test.

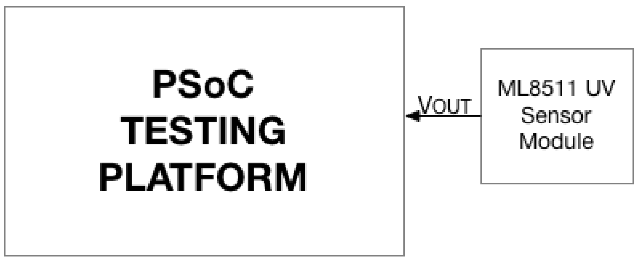


Figure 4.23: Block diagram of UV sensor test using PSoC.

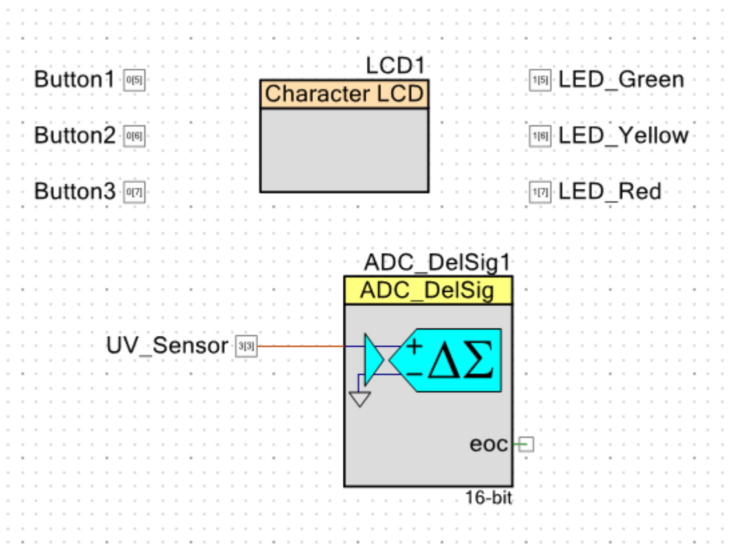


Figure 4.24: PSoC Creator blocks for UV sensor tests.

4.2.4 Magnetometer, Accelerometer and Gyroscope

Like other tests, this test also has the same procedures and code process as the one that was made with the Arduino.

The complete schematic and the source code of this inertial module test can be found on annex IV.

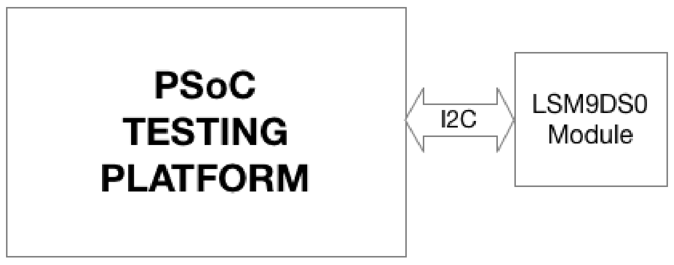


Figure 4.25: Block diagram of magnetometer, accelerometer and gyroscope test using PSoC.

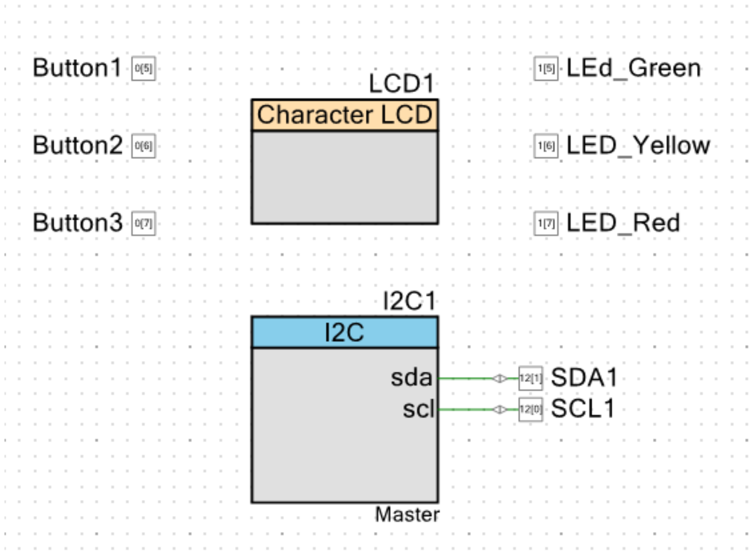


Figure 4.26: PSoC Creator blocks for magnetometer, accelerometer and gyroscope tests.

4.2.5 RTC

The implementation, the procedures and the code processes are equal to the test that was created with the Arduino testing platform.

The complete schematic and the source code of this test are located on annex IV.

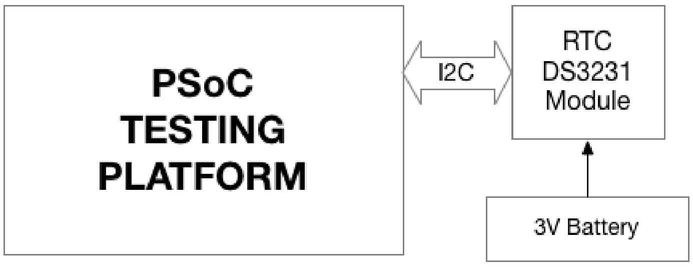


Figure 4.27: Block diagram of RTC test using PSoC.

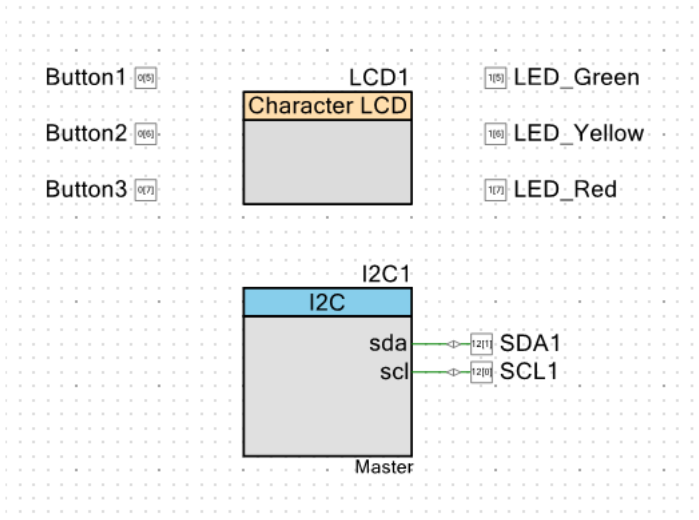


Figure 4.28: PSoC Creator blocks for RTC tests.

4.3 Other Payload Tests

4.3.1 PSOC and Arduino Communication

The communication between Arduino (main controller) and PSoC (payload) is I^{2}C. This test aims to test the Arduino as a master, controlling when its necessary to have communication and the PSoC as a slave, listening and being prepared to send data when requested. This test includes the Arduino testing platform, the PSoC 5LP CY8CKIT-059 and 3 leds connected to PSoC (figure 4.29). Both codes features a time interrupt running every 1 ms.

On figure 4.30 there is the code process diagram of this test.

The complete schematic and the source codes of this test from both controllers are located on annex V.

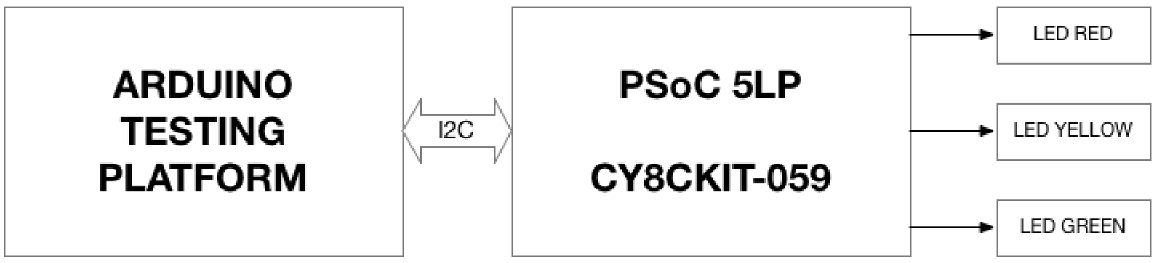


Figure 4.29: Block diagram of I^{2}C communication between Arduino and PSoC.

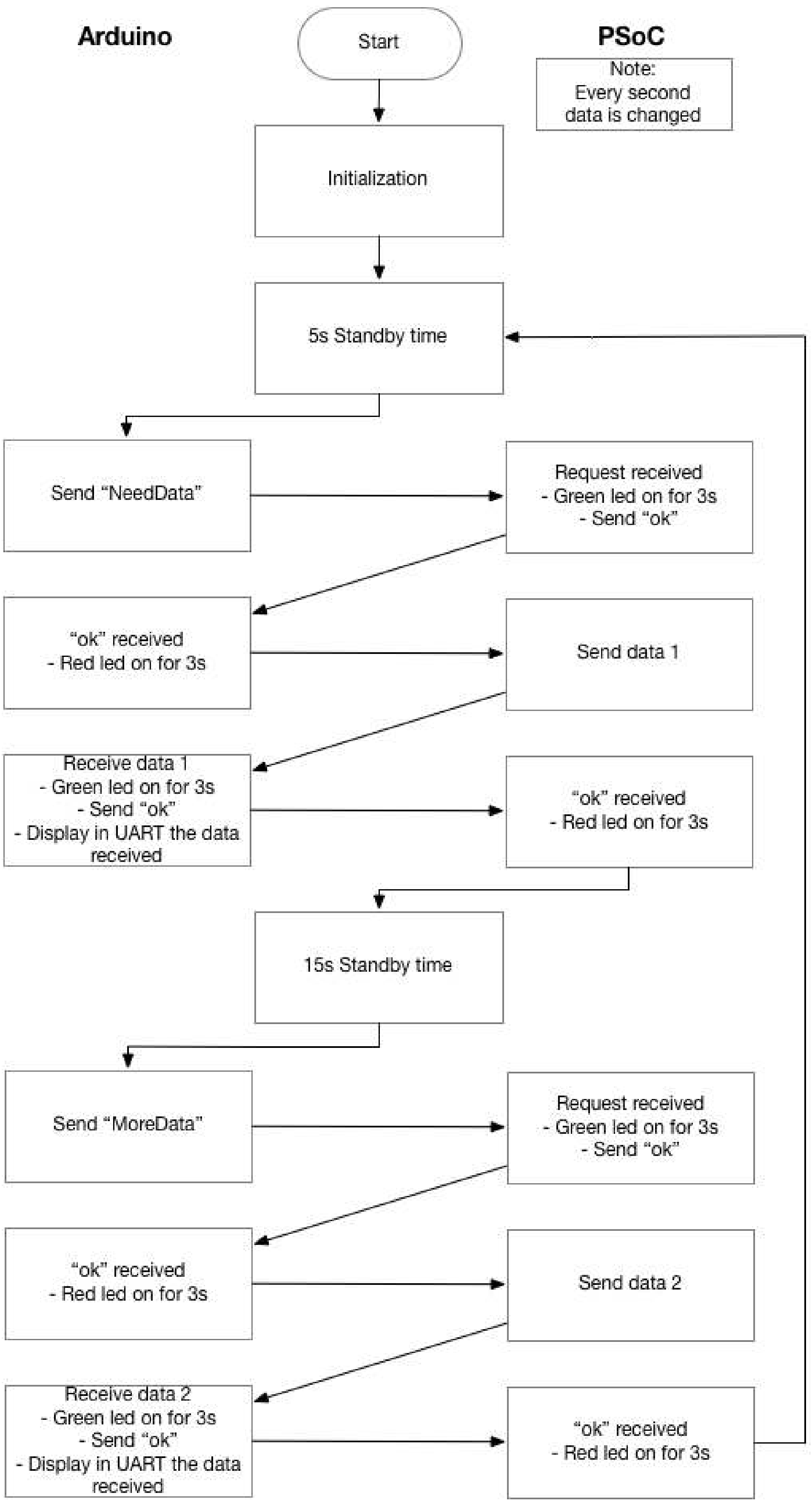

Arduino
Start
PSoC
Note: Every second data is changed
Initialization
5s Standby time
Send "NeedData"
Request received - Green led on for 3s - Send "ok"
"ok" received - Red led on for 3s
Send data 1
Receive data 1 - Green led on for 3s - Send "ok" - Display in UART the data received
"ok" received - Red led on for 3s
15s Standby time
Send "MoreData"
Request received - Green led on for 3s - Send "ok"
"ok" received - Red led on for 3s
Send data 2
Receive data 2 - Green led on for 3s - Send "ok" - Display in UART the data received
"ok" received - Red led on for 3s

Figure 4.30: Code process of I^2C Communication testing between Arduino and PSoC.

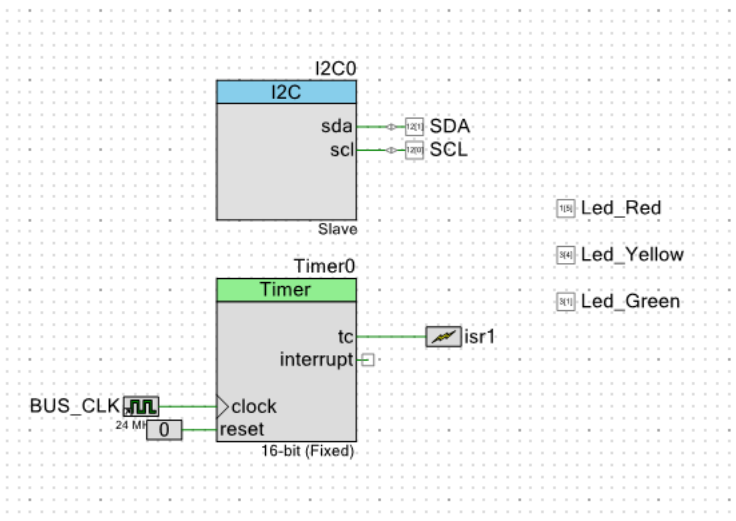


Figure 4.31: PSoC Creator blocks for I^2C Communication testing between Arduino and PSoC.

4.3.2 Voltage Step-up and Battery Charger

The voltage step-up and the battery charger test represent a crucial part of the payload zone. Therefore a test was built with payload similarity which was the step-up connected to battery charger (figure 4.32). The batteries for this test include a backup battery equal to the one found in the payload and other battery as the main power supply with the same chemistry technology (Li-Ion).

The complete schematic of this test is located in annex V.

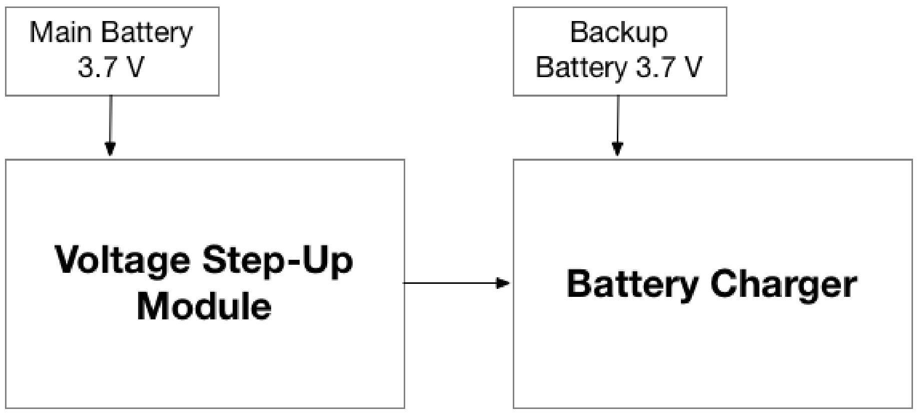


Figure 4.32: Block diagram of Step-Up and Charger test.

This test follows the procedures presented below.

Procedures:

1. Connect the backup battery and then the main battery.

2. Turn on the system and start the chronometer. Register the input, output, backup battery voltages as well as the output voltage of the first module which is the input for the charger.

3. Keep watching the circuit until the charger's orange led turn off and the green led turn on. In that moment register the voltages mentioned above and register the time it takes to charge if the backup battery is different from 4.2 V (battery charged).

4. Repeat all process for different main voltages (3.6, 4.2, 3.0) and different backup battery voltages (3.7, 4.2, 3.0)

4.4 Payload Board Tests

4.4.1 Operation

The CTSAT-1's payload board test presents the compilation of every module and sensor tested before with the inclusion of some more electronics components, like the flash memory and associated components (logic level translator and voltage regulator), and the backup battery.

The schematic can be found on annex VI, although the source codes will not be provided due to thousand of lines of code.

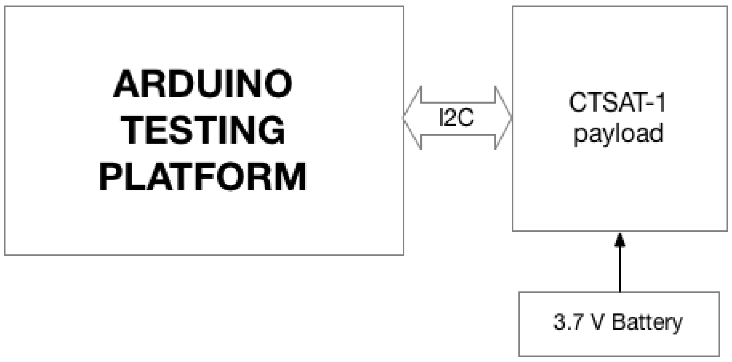


Figure 4.33: Block diagram of CTSAT-1 payload test.

For this test, it was only tested the ability of sending messages and data to the Arduino testing platform.

So the procedures it's somehow similar with the procedures of I^2C communication test, but without some transmission flags that could make the communication fail.

4.4.2 Power Consumption

On figure 4.34 is the block diagram for the power consumption test, that will evaluate the current energy spent by the payload board.

The procedures for this test are not very specific, but the test measurements must follow the following configurations.

1. Standby.
2. Sensors / Modules reading.
3. Sending data through I^2C.
4. Receive data through I^2C.

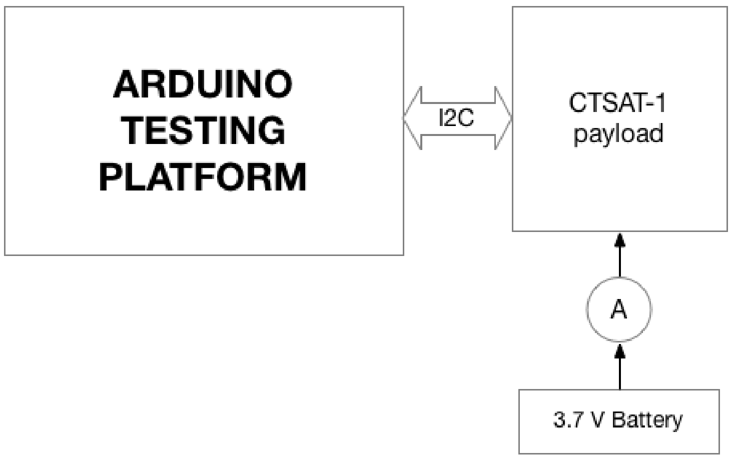


Figure 4.34: Block diagram of CTSAT-1 payload power consumption test.

Chapter 5

Measurement Results and Analysis

The measurement results provided by the mentioned tests of chapter 4 are exposed here in this chapter.

5.1 Testing Payload Components Results

For the payload components tests, the results represented a step to validation of the component's operations.

5.1.1 RTD Temperature

For the RTD temperature tests, almost 360 measurements were retrieved from each RTD individual tests during half an hour each test.

The ice and the hot air allowed to watch the sensor responsiveness through the high temperature variations.

5.1.1.1 Experiment 1 - USB

From this experiment, only done with the Arduino testing platform (figure 5.1), the data was picked up through UART. After having a file with all the data, a part of the file is converted to temperature with the help of equations provided on chapter 3. The other part of the file have the temperatures acquired through mapping inside Arduino.

The generated chart (figure 5.2) overlaps this two methods results.

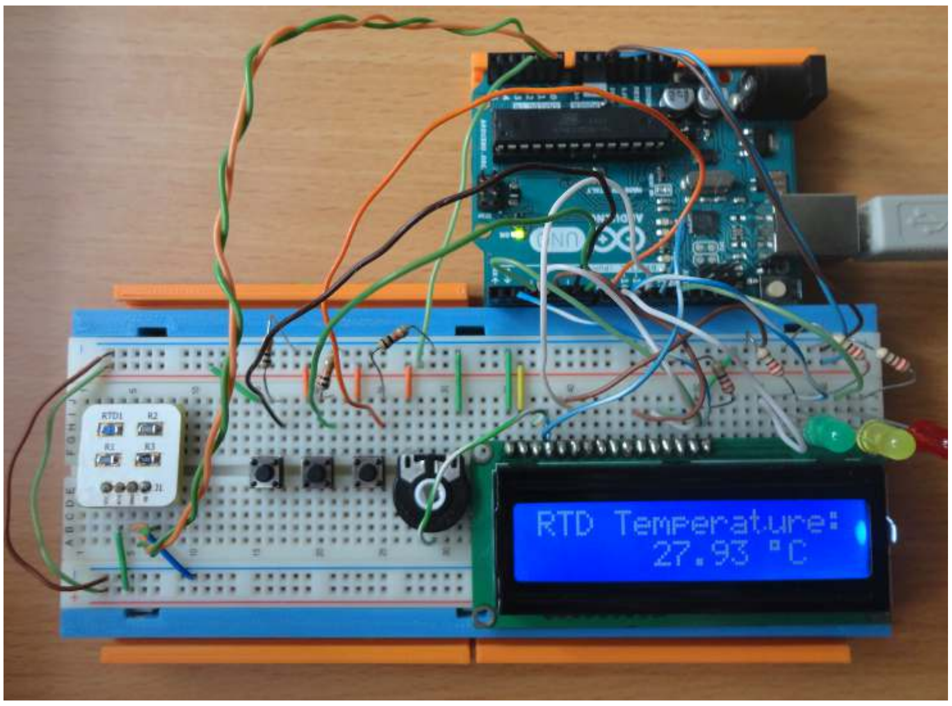


Figure 5.1: RTD experiment 1 testing on Arduino.

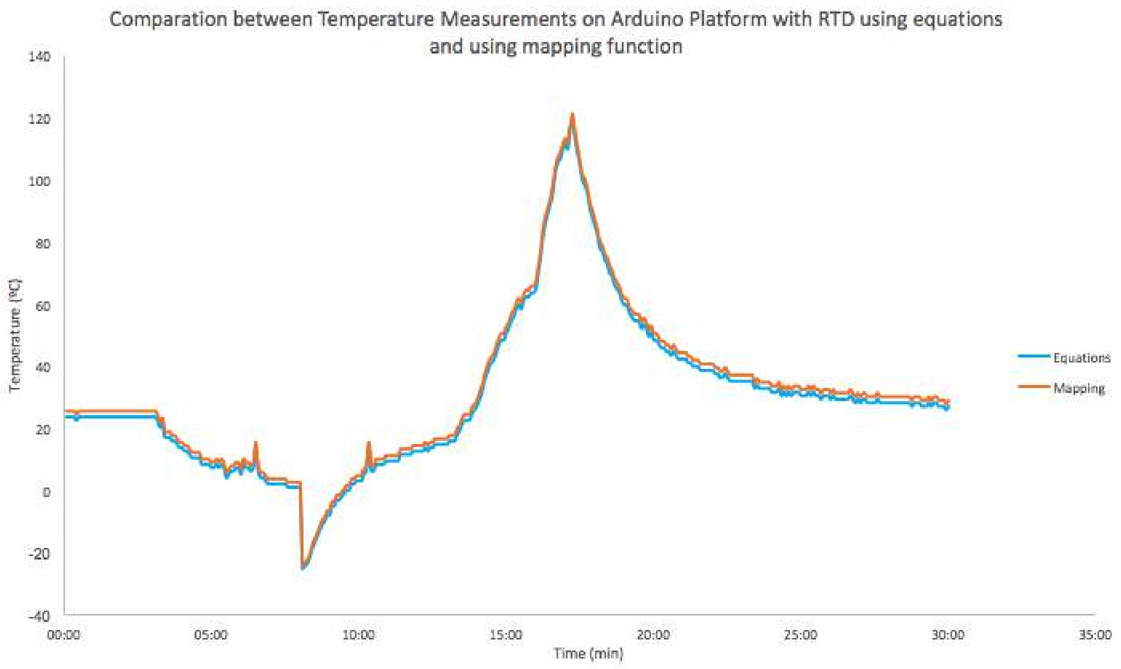
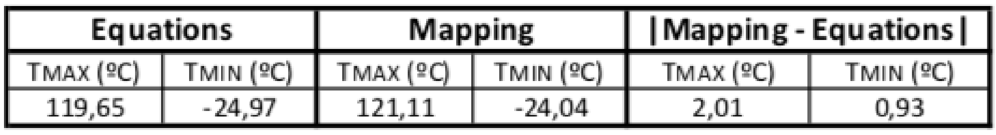


Figure 5.2: Chart of comparation between temperature measurements on Arduino platform with RTD using equations and using mapping function - experiment 1.



Equations		Mapping		|Mapping - Equations|	
TMAX (ºC)	TMIN (ºC)	TMAX (ºC)	TMIN (ºC)	TMAX (ºC)	TMIN (ºC)
119,65	-24,97	121,11	-24,04	2,01	0,93

Table 5.1: Table with maximums and minimums of RTD experiment 1 test on Arduino platform using equations and using mapping function.

The two methods, equations and mapping, produce almost all the same data values with slight variation. Despite having this variation, this proves the linearity of the sensor which is present in the component's datasheet.

The temperature test produced a maximum of 2,01 ºC between the two methods. That difference can be explained with the approximation of the mapping function.

5.1.1.2 Experiment 2 SD Card

On this experiment the data maintained the same data format that the with USB connection. On the SD Card, it was recorded the resistance of the RTD plus the temperature that was converted through mapping inside the controller.

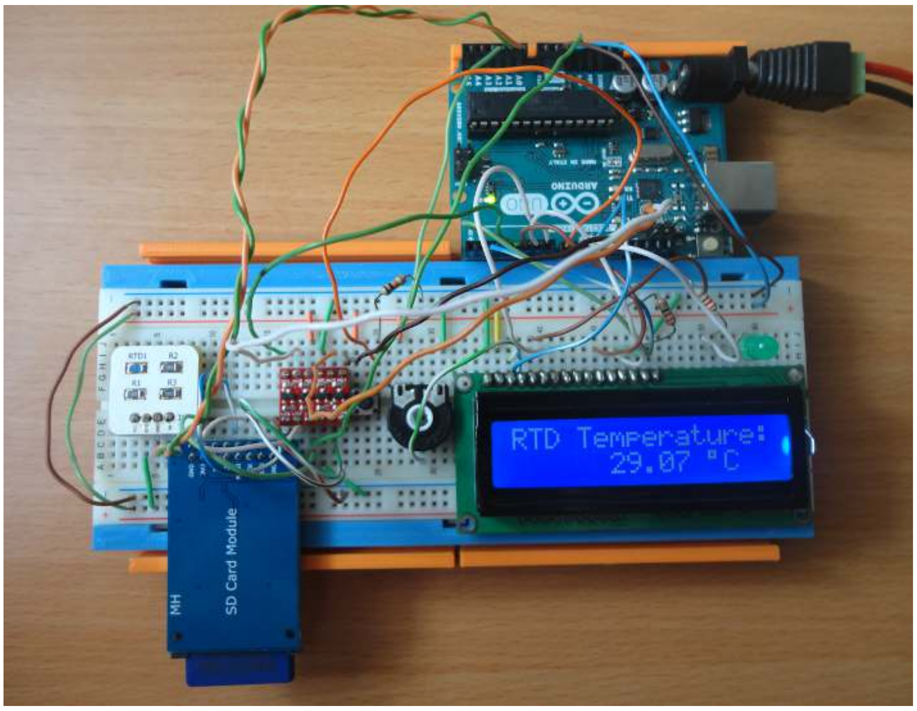


Figure 5.3: RTD experiment 2 testing on Arduino.

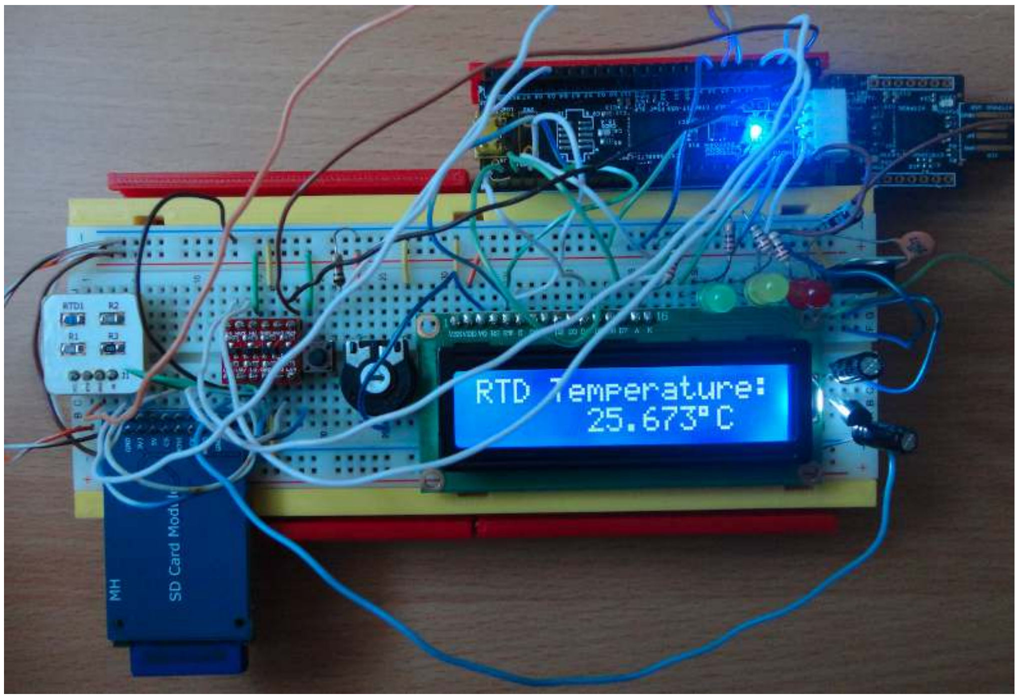


Figure 5.4: RTD testing on PSoC.

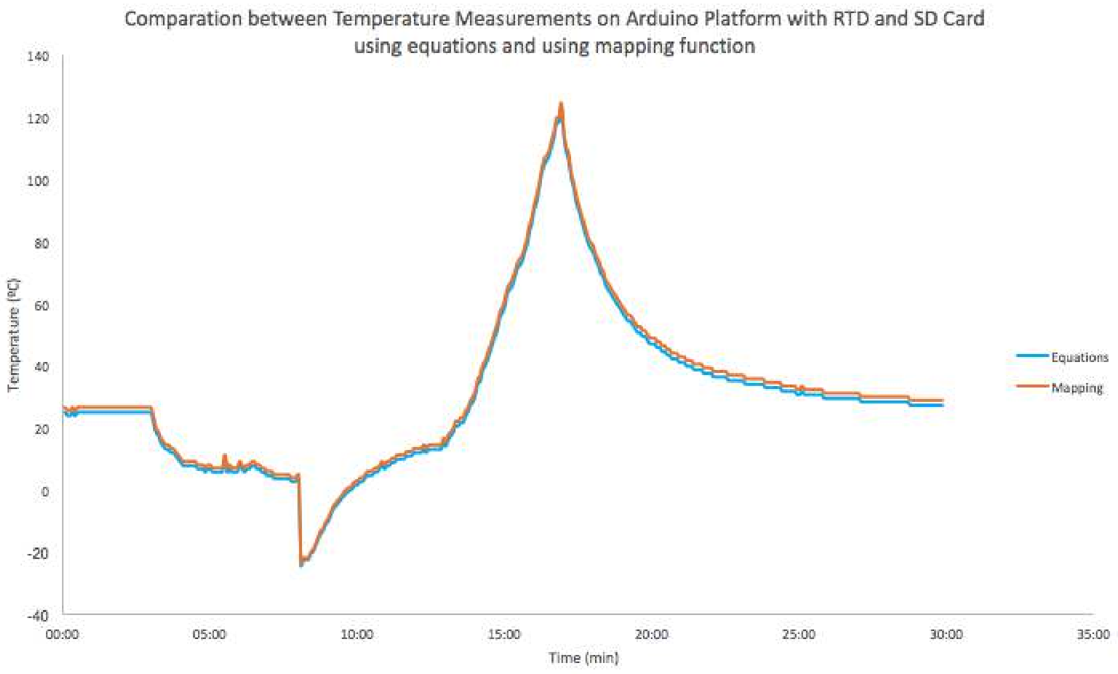


Figure 5.5: Chart of comparation between temperature measurements on Arduino platform with RTD and SD card using equations and using mapping function - experiment 2.

Equations		Mapping		|Mapping - Equations|	
TMAX (ºC)	TMIN (ºC)	TMAX (ºC)	TMIN (ºC)	TMAX (ºC)	TMIN (ºC)
123,01	-24,36	124,40	-23,42	2,01	0,94

Table 5.2: Table with maximums and minimums of RTD experiment 2 test on Arduino platform using equations and using mapping function.

On the figure 5.5, it can be seen that with SD Card the shape of the chart remains practically the same.

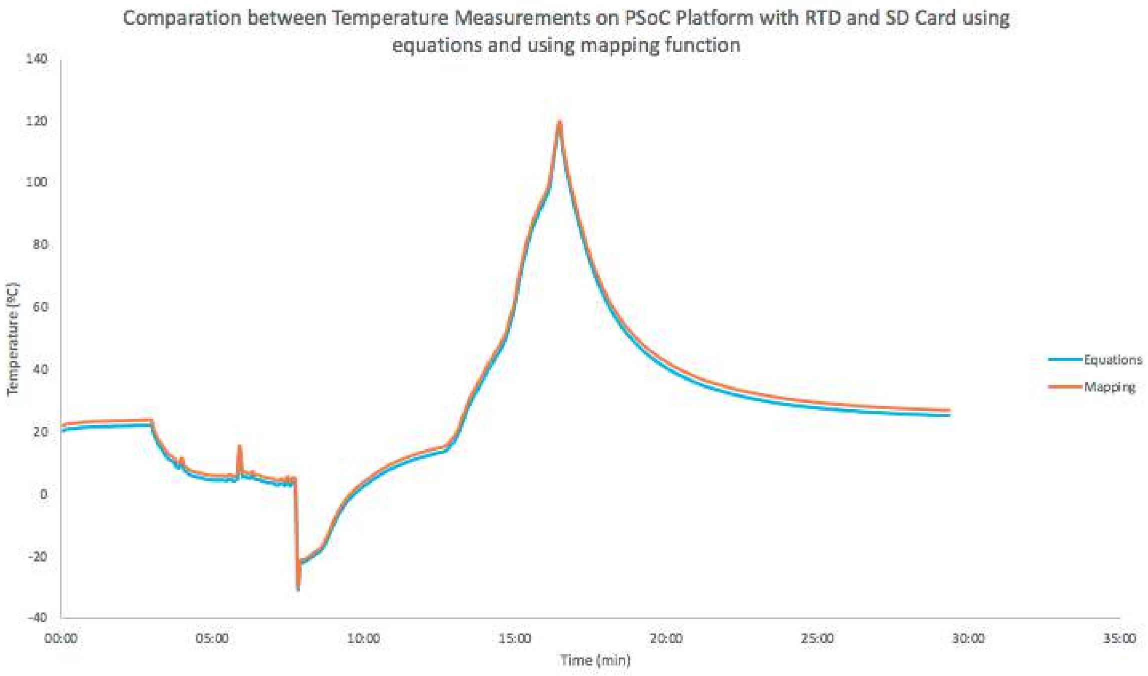


Figure 5.6: Chart of comparation between temperature measurements on PSoC platform with RTD and SD card using equations and using mapping function.

Equations		Mapping		|Mapping - Equations|	
TMAX (ºC)	TMIN (ºC)	TMAX (ºC)	TMIN (ºC)	TMAX (ºC)	TMIN (ºC)
118,08	-29,96	119,57	-29,17	2,00	0,79

Table 5.3: Table with maximums and minimums of RTD test on PSoC platform using equations and using mapping function.

The Arduino and PSoC produced similar results, even in the difference between the method of equations and method of mapping.

The sharp peak in the negative temperatures seen in all charts are due to the direct ice contact (without the cloth) with the RTD.

5.1.2 Digital Temperature

In the digital temperature sensor tests results, the SD Card was used again to store the temperature values.

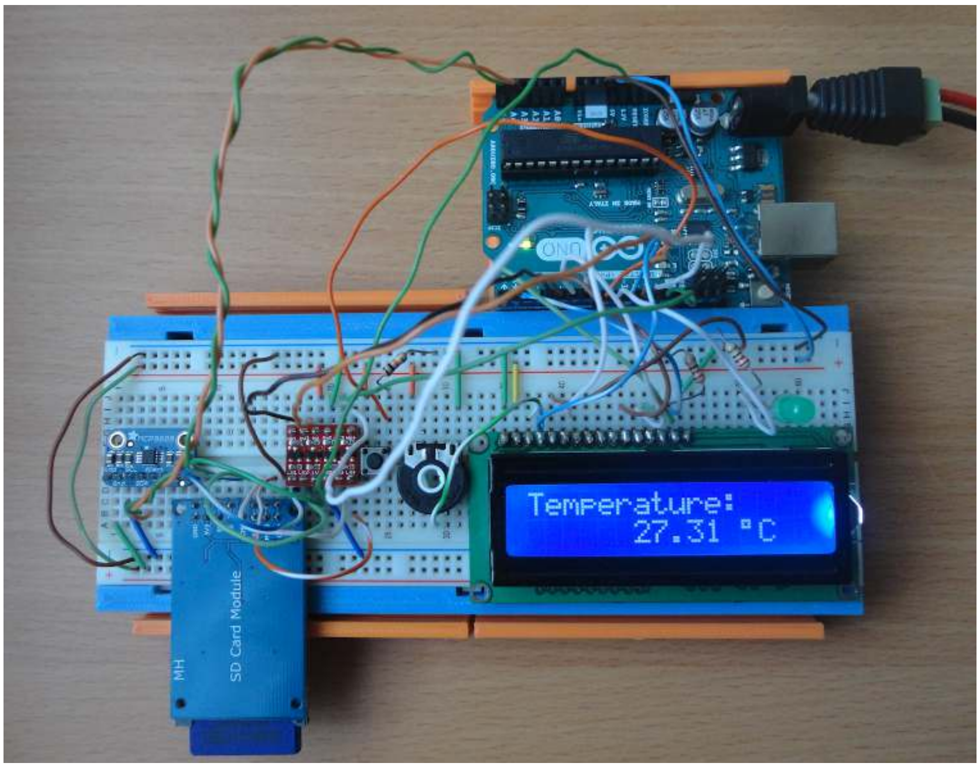


Figure 5.7: Digital temperature sensor testing on Arduino.

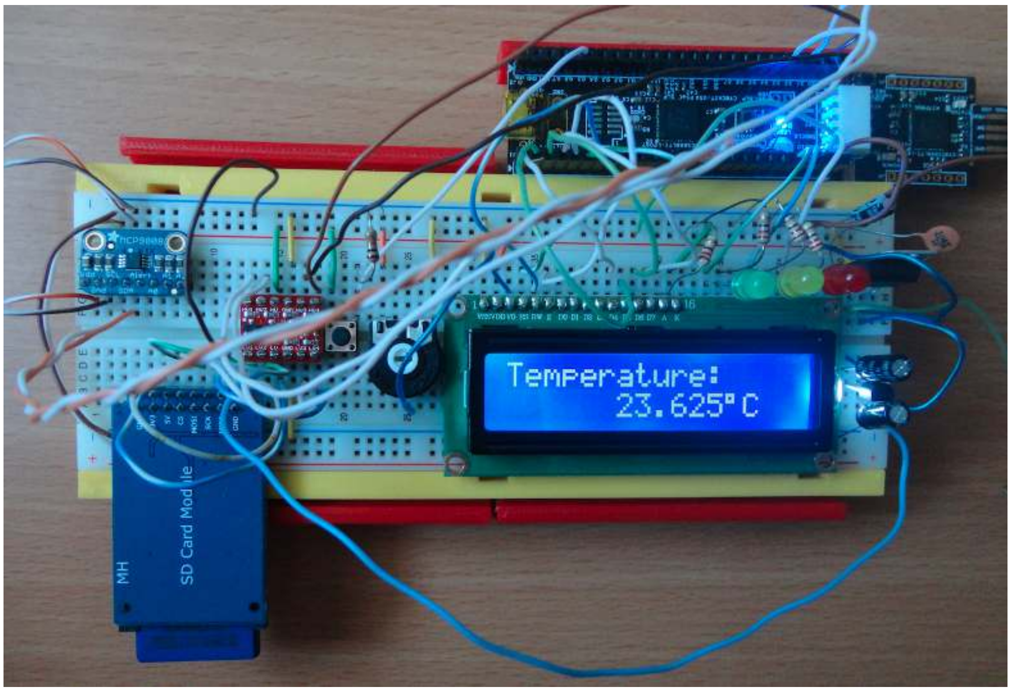


Figure 5.8: Digital temperature sensor testing on PSoC.

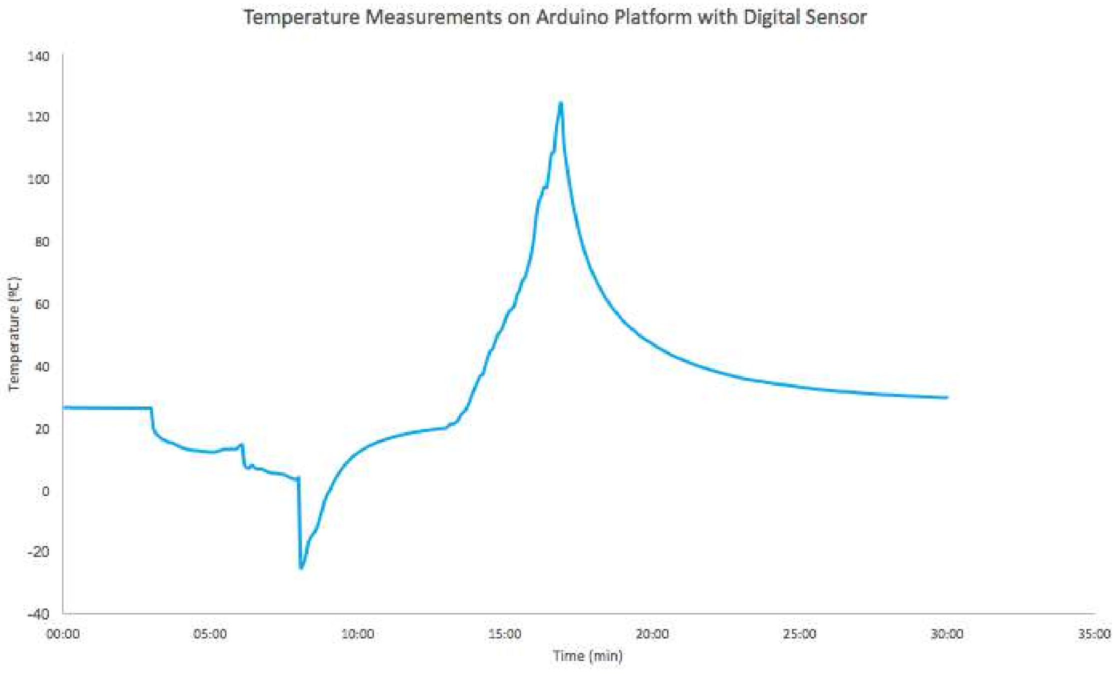


Figure 5.9: Chart of temperature measurements on Arduino platform with digital sensor.

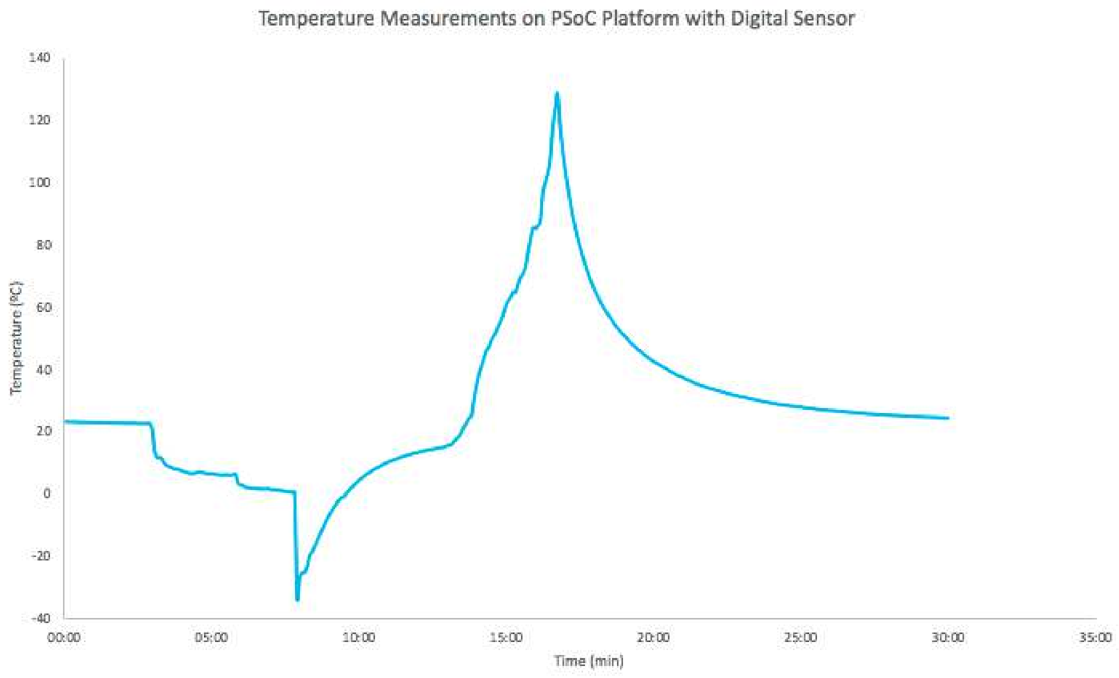


Figure 5.10: Chart of temperature measurements on PSoC platform with digital sensor.

In the figures 5.9 and 5.10 is showed the testing with the Arduino testing platform and the PSoC testing platform, respectively.

This tests followed the same procedures as the ones for the RTD, as such resulting in the same graphical data response.

With the Arduino the temperature reached a maximum of 124,06 ºC and a minimum of -25,12 ºC, with the PSoC the temperature achieved a maximum of 128,88 ºC and a minimum of -33,44 ºC.

5.1.3 UV Sensor

The UV sensor tests(figures 5.11 and 5.12) were only possible one days without to much clouds in the sky and without rain, the day must have moments of sun direct light.

When the UV sensor test was done with Arduino testing platform, the UV index maximum for that day was 8 with a temperature maximum of 34 ºC. On the test with PSoC the UV index maximum was 6 with a maximum temperature of 28ºC. This information was provided by the Instituto Português do Mar e da Atmosfera (IPMA).

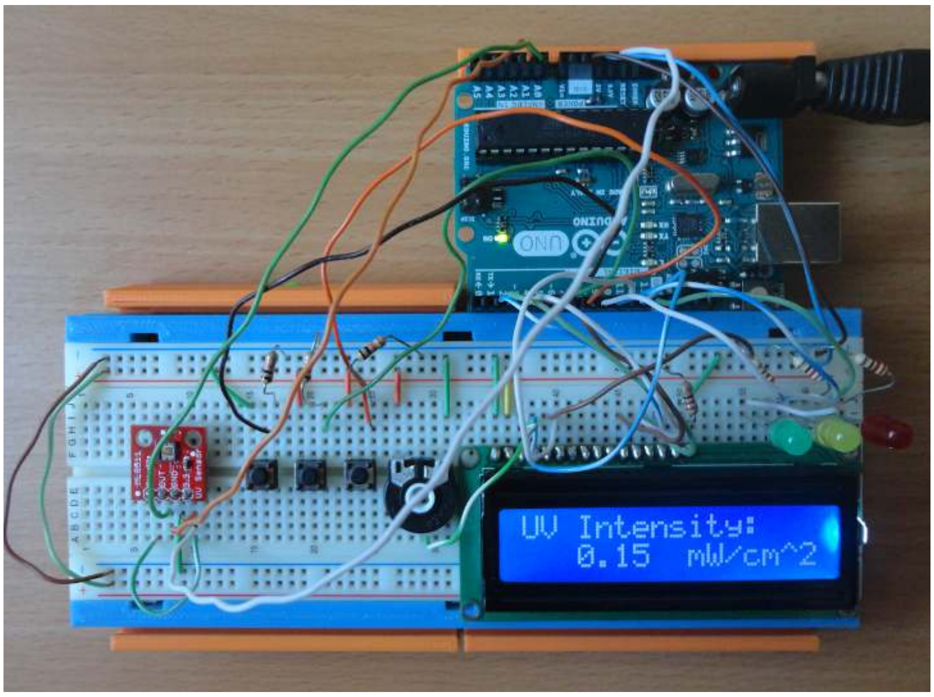


Figure 5.11: UV sensor testing on Arduino.

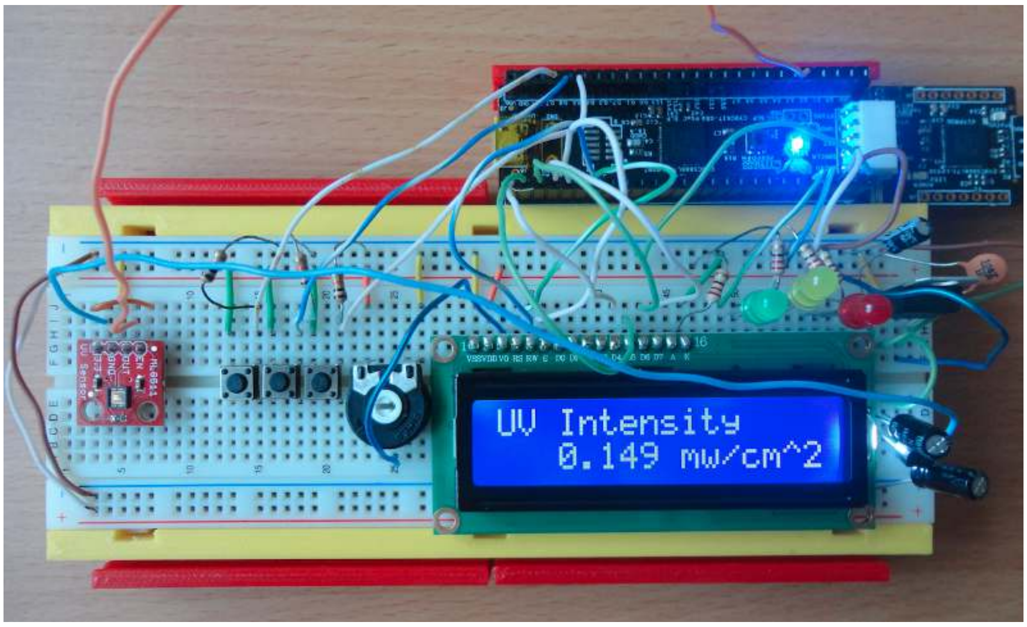


Figure 5.12: UV sensor testing on PSoC.

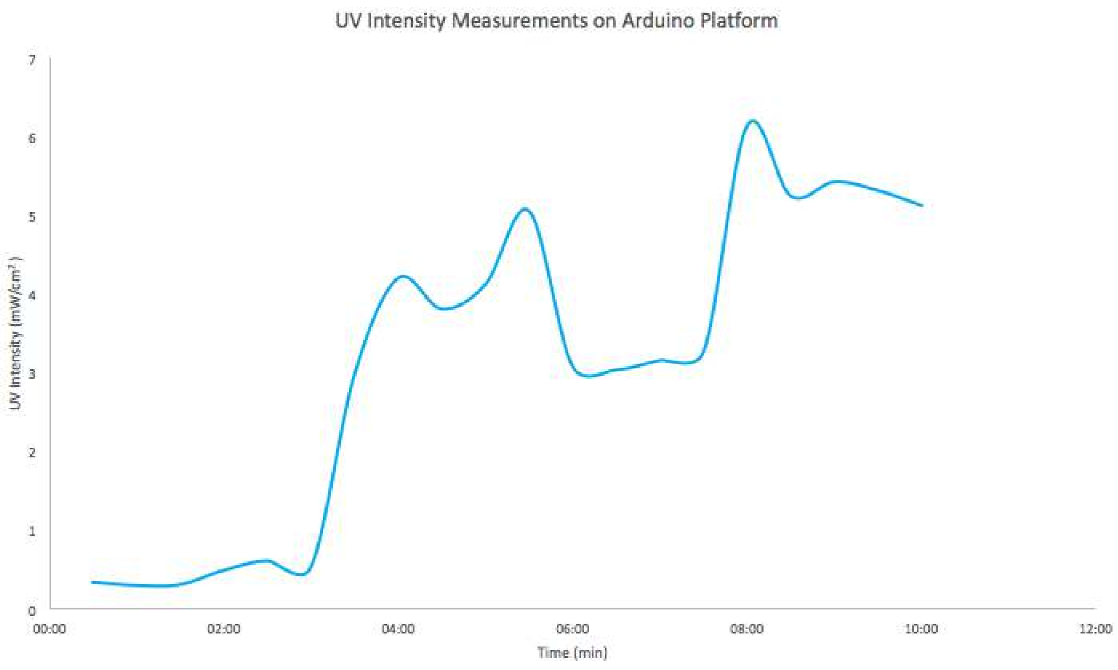


Figure 5.13: Chart of UV intensity measurements on Arduino platform.

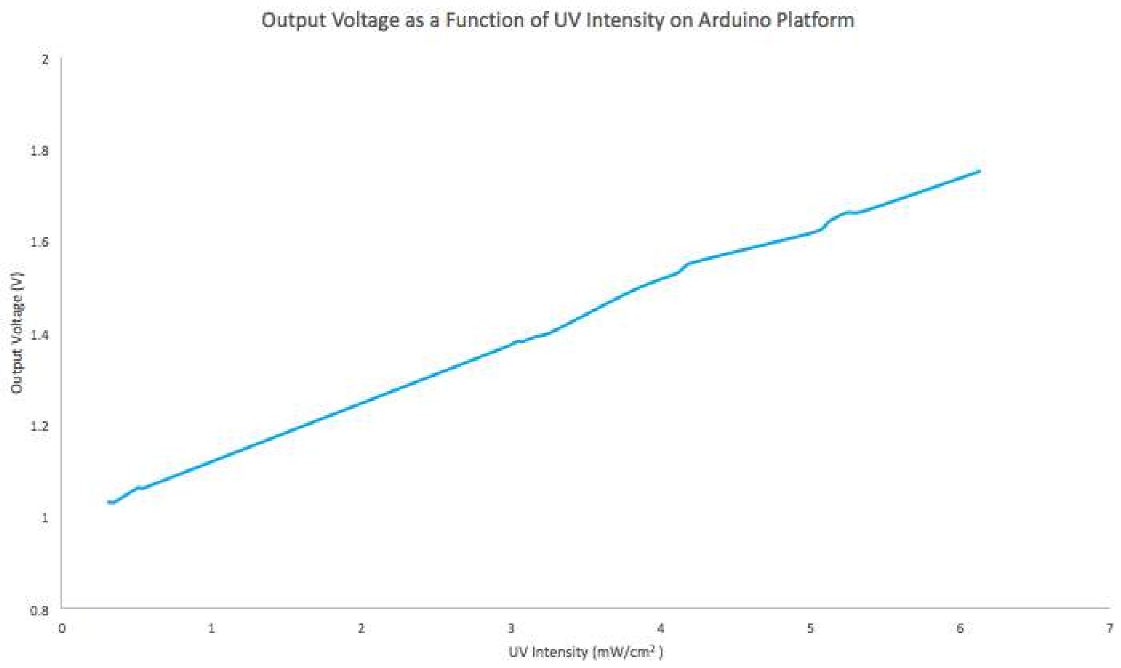
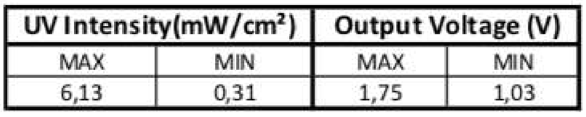


Figure 5.14: Chart of output voltage as a function of UV intensity on Arduino Platform



UV Intensity(mW/cm^2)		Output Voltage (V)	
MAX	MIN	MAX	MIN
6,13	0,31	1,75	1,03

Table 5.4: Table with maximums and minimums of output data from UV sensor test on Arduino platform using equations and using mapping function.

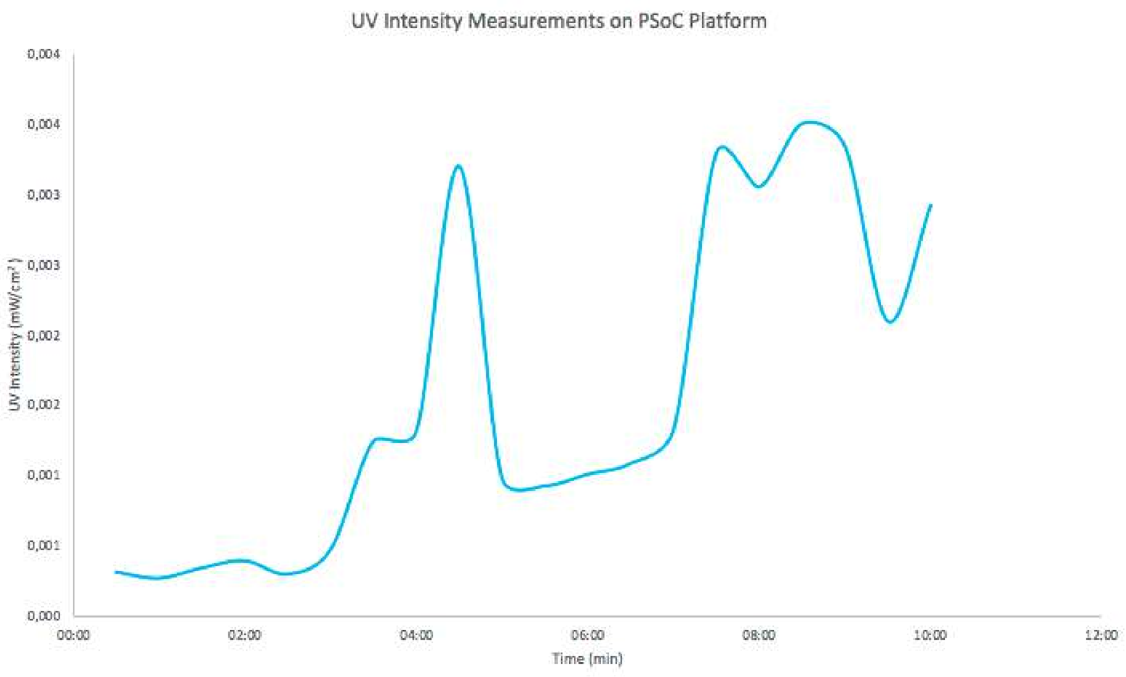


Figure 5.15: Chart of UV intensity measurements on PSoC platform.

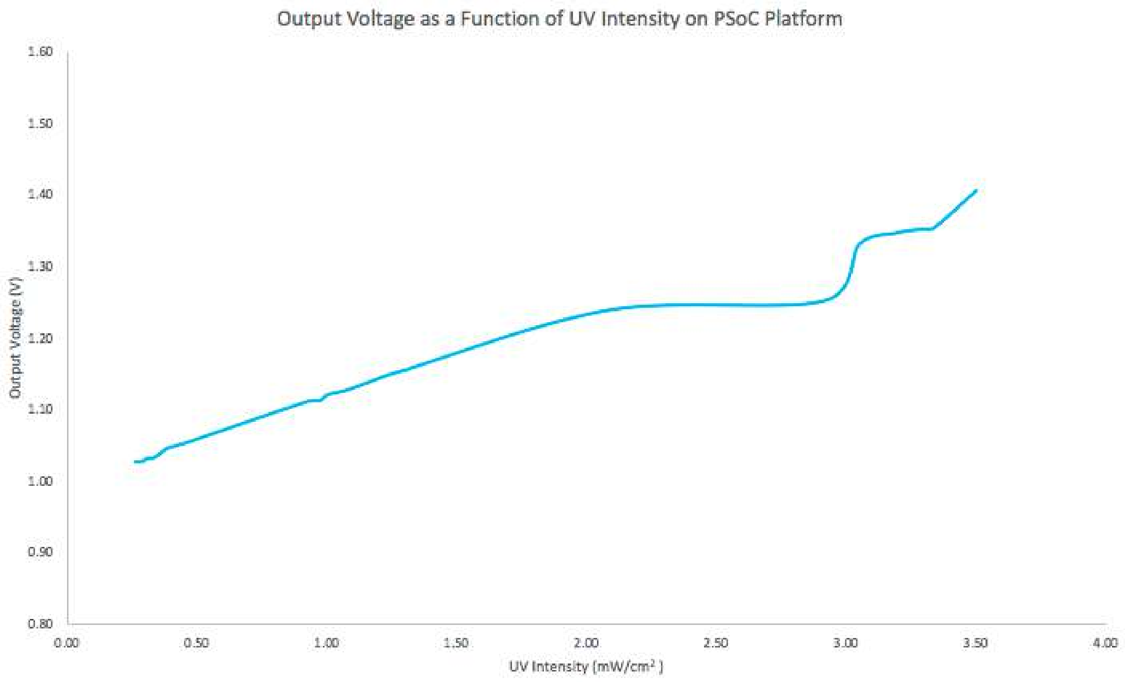


Figure 5.16: Chart of output voltage as a function of UV intensity on PSoC Platform

UV Intensity(mW/cm²)		Output Voltage (V)	
MAX	MIN	MAX	MIN
3,50	0,26	1,41	1,03

Table 5.5: Table with maximums and minimums of output data from UV sensor test on PSoC platform using equations and using mapping function.

As it is possible to watch on figures 5.13 and 5.15 the graph shape is similar where is possible to see the highlighted areas of shadow zone and zone without direct sunlight until the minute three and areas of ultraviolet radiation reception.

In terms of sensor's linearity. on the Arduino (figure 5.14) the sensor proves to have a very good linearity, which confirms the sensor's datasheet. On the PSoC (figure 5.16) in the beginning the sensor showed a good linearity but towards the end of the test the data showed a slightly deviation from a good linearity.

Due to the Arduino's test was made in a hotter day and consequently more ultraviolet radiation reached to Earth's surface, the maximum UV intensity was 6,13 mW/cm^2 versus 3,50 mW/cm^2 on the PSoC testing platform (tables 5.4 and 5.5).

5.1.4 Magnetometer, Accelerometer and Gyroscope

The tests with the inertial module were made following the positions of figure 4.13. This tests were initially made with the raw data from the sensor, but later they were changed to the correct form of treated data to be easier to understand.

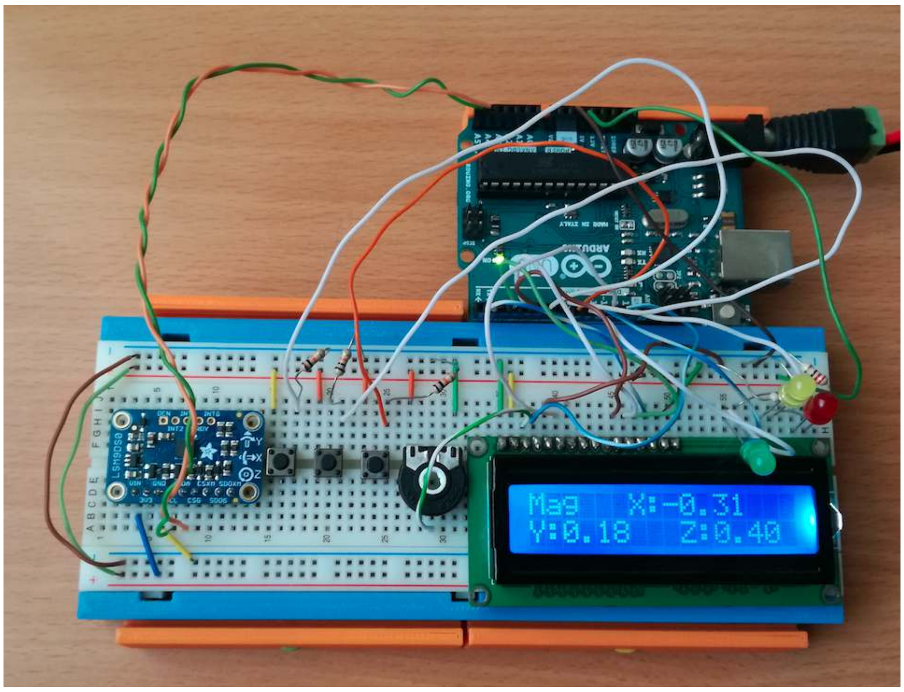


Figure 5.17: Magnetometer testing on Arduino.

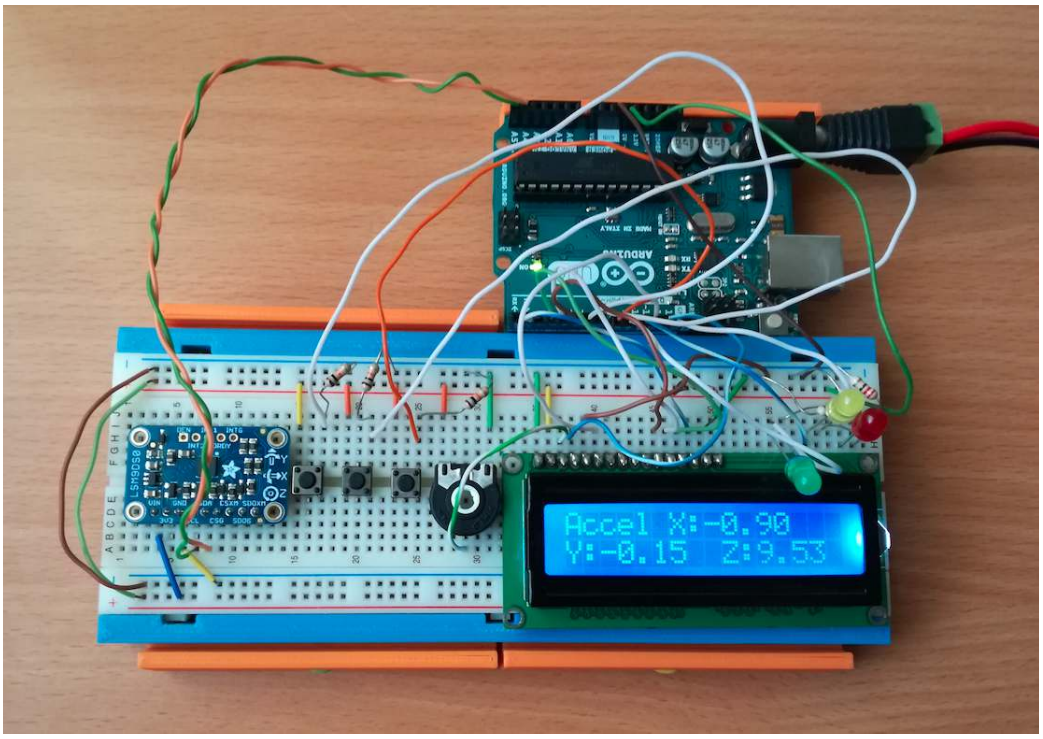


Figure 5.18: Accelerometer testing on Arduino.

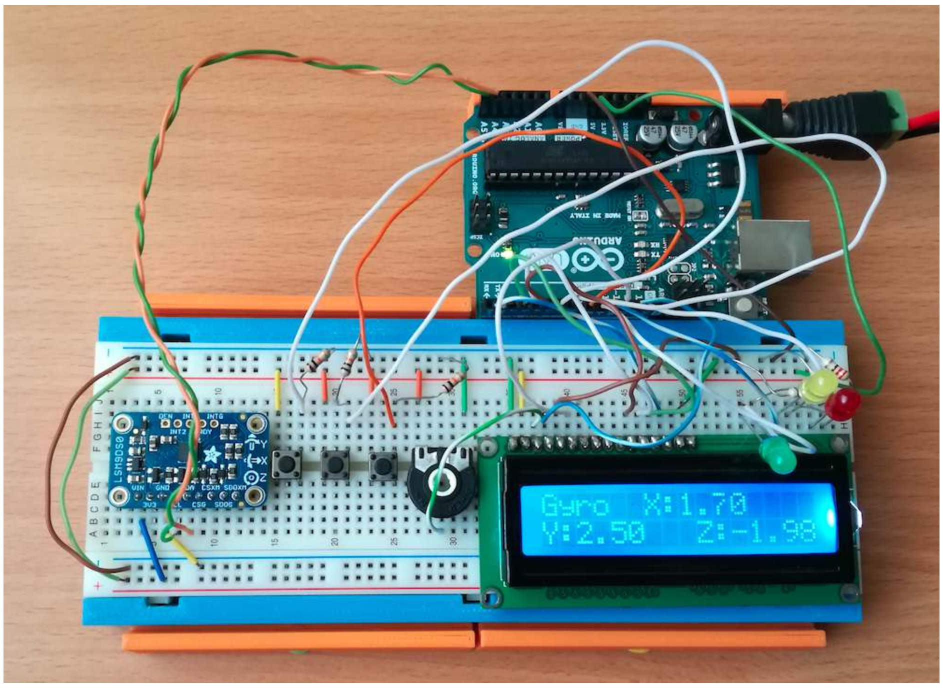


Figure 5.19: Gyroscope testing on Arduino.

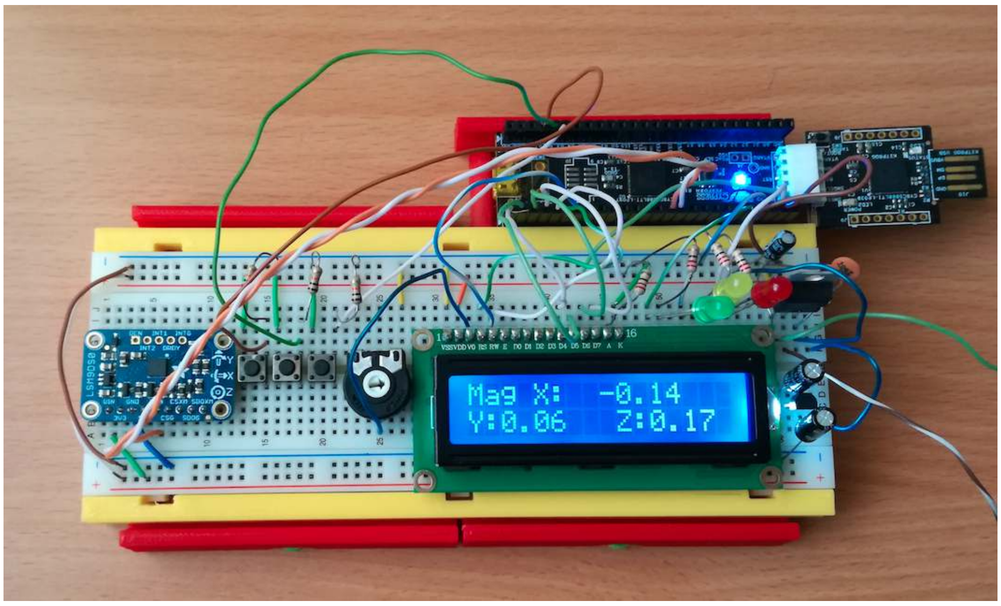


Figure 5.20: Magnetometer testing on PSoC.

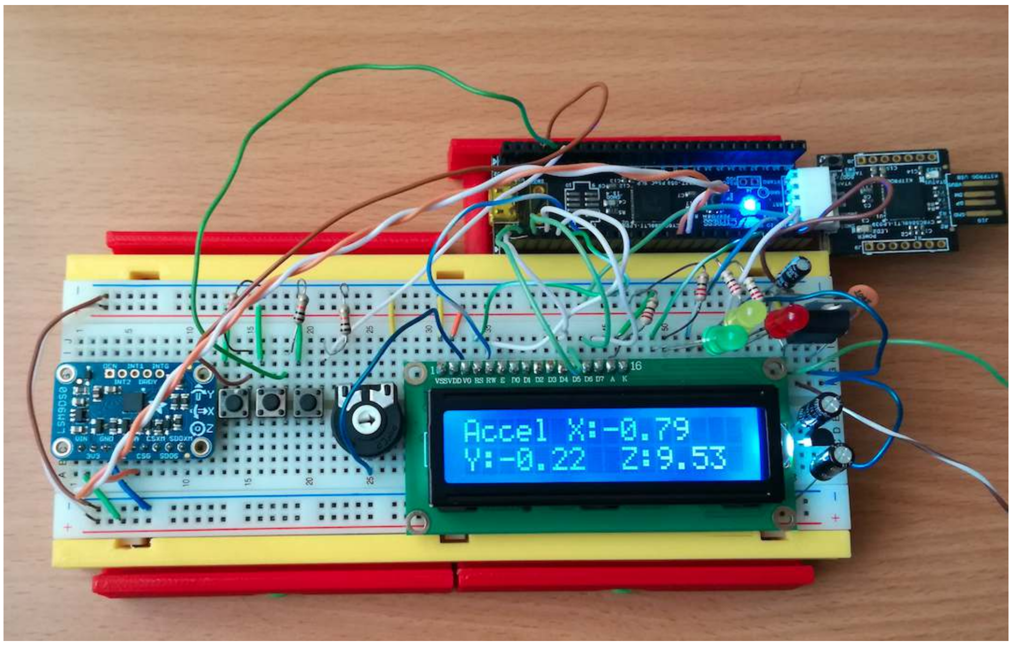


Figure 5.21: Accelerometer testing on PSoC.

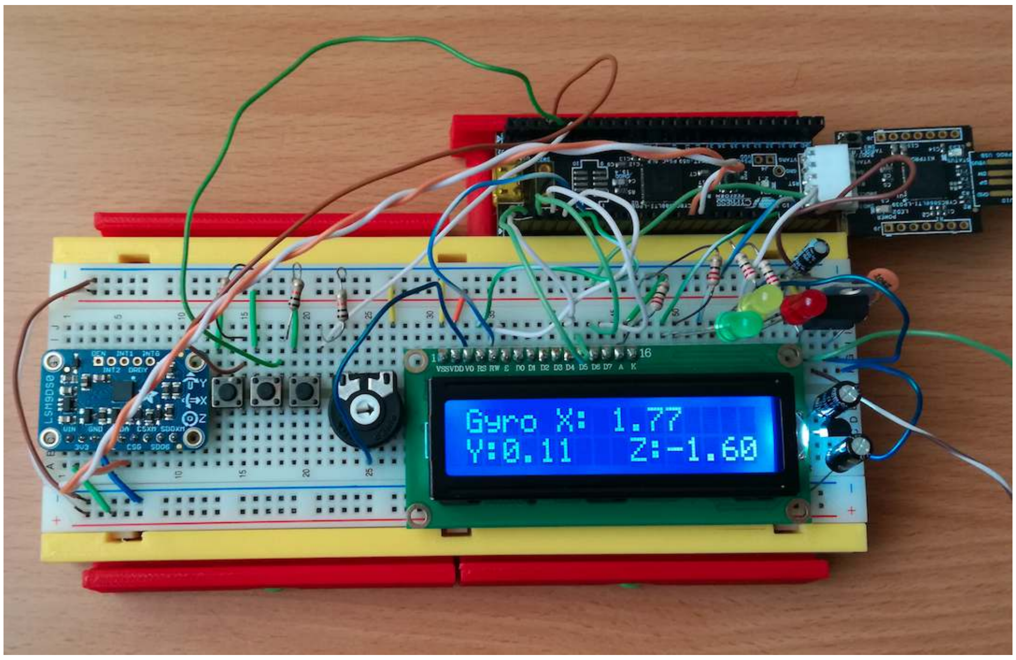
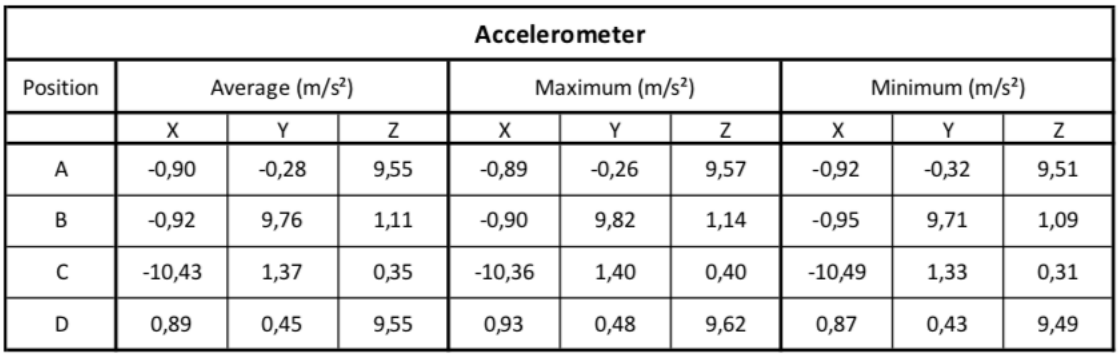
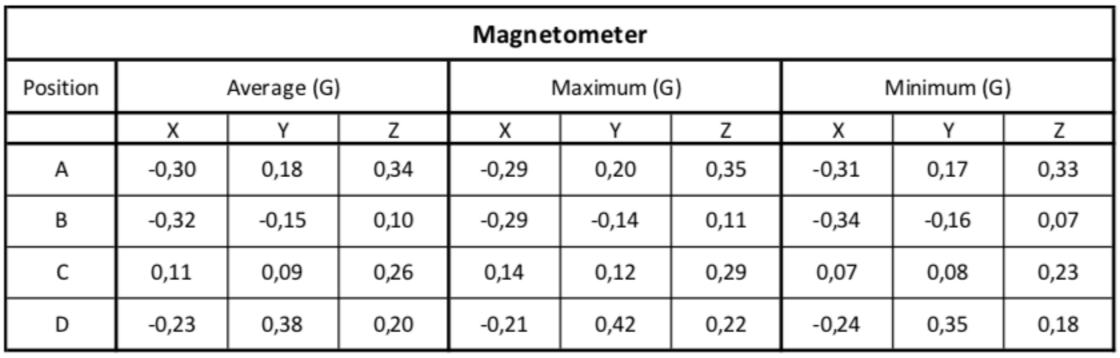


Figure 5.22: Gyroscope testing on PSoC.



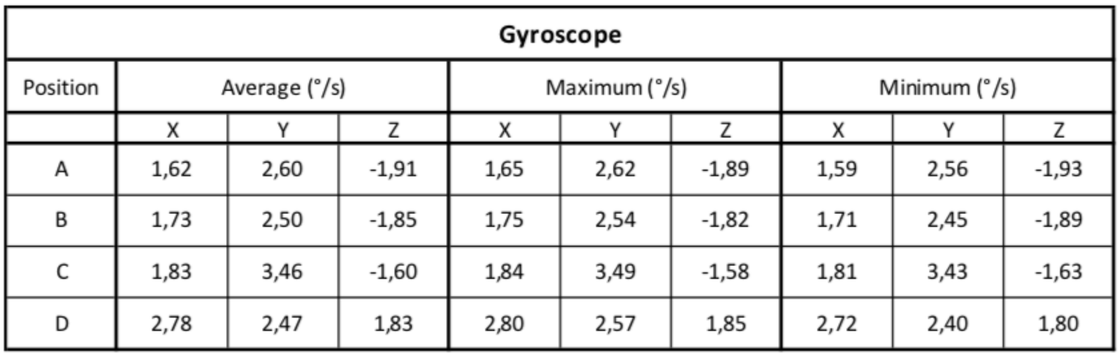
Accelerometer									
Position	Average (m/s²)			Maximum (m/s²)			Minimum (m/s²)		
	X	Y	Z	X	Y	Z	X	Y	Z
A	-0,90	-0,28	9,55	-0,89	-0,26	9,57	-0,92	-0,32	9,51
B	-0,92	9,76	1,11	-0,90	9,82	1,14	-0,95	9,71	1,09
C	-10,43	1,37	0,35	-10,36	1,40	0,40	-10,49	1,33	0,31
D	0,89	0,45	9,55	0,93	0,48	9,62	0,87	0,43	9,49

Table 5.6: Table with the averages, maximums and minimums of accelerometer test on Arduino platform in the different positions tested.



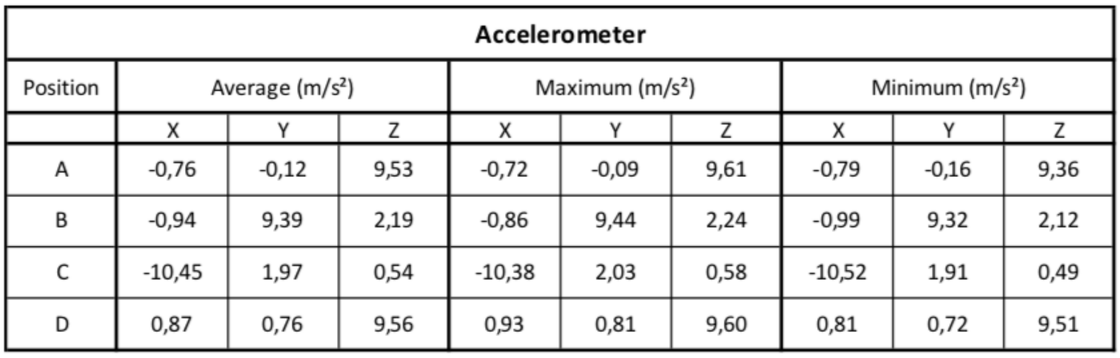
Magnetometer									
Position	Average (G)			Maximum (G)			Minimum (G)		
	X	Y	Z	X	Y	Z	X	Y	Z
A	-0,30	0,18	0,34	-0,29	0,20	0,35	-0,31	0,17	0,33
B	-0,32	-0,15	0,10	-0,29	-0,14	0,11	-0,34	-0,16	0,07
C	0,11	0,09	0,26	0,14	0,12	0,29	0,07	0,08	0,23
D	-0,23	0,38	0,20	-0,21	0,42	0,22	-0,24	0,35	0,18

Table 5.7: Table with the averages, maximums and minimums of magnetometer test on Arduino platform in the different positions tested.



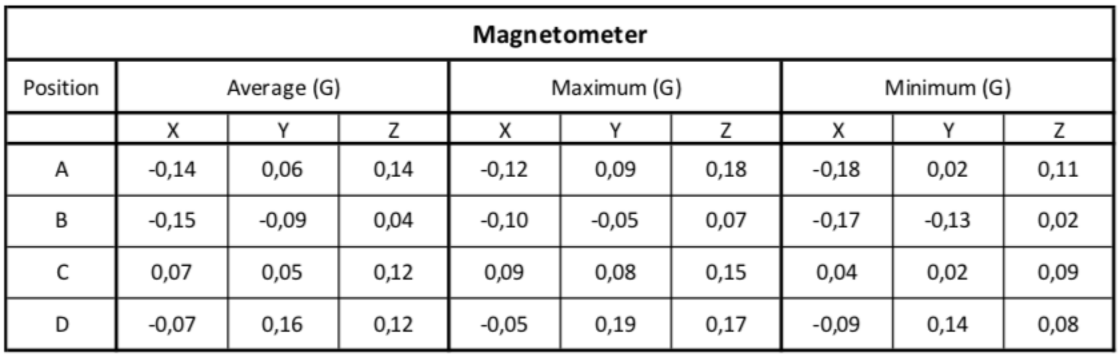
Gyroscope									
Position	Average (°/s)			Maximum (°/s)			Minimum (°/s)		
	X	Y	Z	X	Y	Z	X	Y	Z
A	1,62	2,60	-1,91	1,65	2,62	-1,89	1,59	2,56	-1,93
B	1,73	2,50	-1,85	1,75	2,54	-1,82	1,71	2,45	-1,89
C	1,83	3,46	-1,60	1,84	3,49	-1,58	1,81	3,43	-1,63
D	2,78	2,47	1,83	2,80	2,57	1,85	2,72	2,40	1,80

Table 5.8: Table with the averages, maximums and minimums of gyroscope test on Arduino platform in the different positions tested.



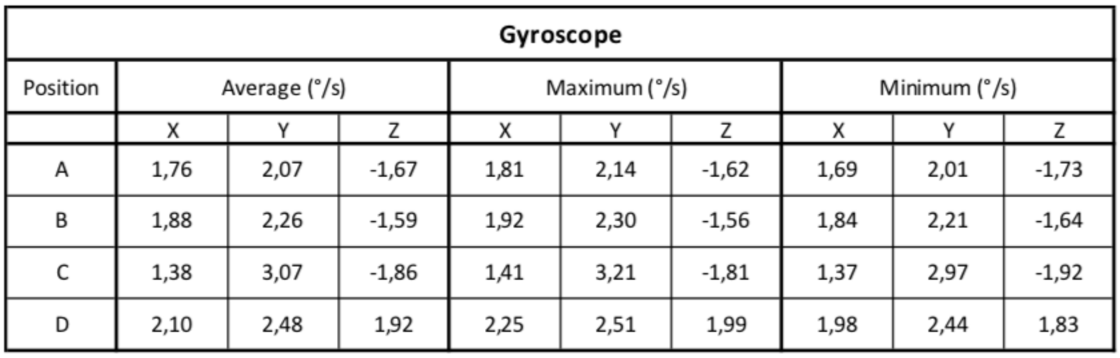
Accelerometer									
Position	Average (m/s²)			Maximum (m/s²)			Minimum (m/s²)		
	X	Y	Z	X	Y	Z	X	Y	Z
A	-0,76	-0,12	9,53	-0,72	-0,09	9,61	-0,79	-0,16	9,36
B	-0,94	9,39	2,19	-0,86	9,44	2,24	-0,99	9,32	2,12
C	-10,45	1,97	0,54	-10,38	2,03	0,58	-10,52	1,91	0,49
D	0,87	0,76	9,56	0,93	0,81	9,60	0,81	0,72	9,51

Table 5.9: Table with the averages, maximums and minimums of accelerometer test on PSoC platform in the different positions tested.



Magnetometer									
Position	Average (G)			Maximum (G)			Minimum (G)		
	X	Y	Z	X	Y	Z	X	Y	Z
A	-0,14	0,06	0,14	-0,12	0,09	0,18	-0,18	0,02	0,11
B	-0,15	-0,09	0,04	-0,10	-0,05	0,07	-0,17	-0,13	0,02
C	0,07	0,05	0,12	0,09	0,08	0,15	0,04	0,02	0,09
D	-0,07	0,16	0,12	-0,05	0,19	0,17	-0,09	0,14	0,08

Table 5.10: Table with the averages, maximums and minimums of magnetometer test on PSoC platform in the different positions tested.



Gyroscope									
Position	Average (°/s)			Maximum (°/s)			Minimum (°/s)		
	X	Y	Z	X	Y	Z	X	Y	Z
A	1,76	2,07	-1,67	1,81	2,14	-1,62	1,69	2,01	-1,73
B	1,88	2,26	-1,59	1,92	2,30	-1,56	1,84	2,21	-1,64
C	1,38	3,07	-1,86	1,41	3,21	-1,81	1,37	2,97	-1,92
D	2,10	2,48	1,92	2,25	2,51	1,99	1,98	2,44	1,83

Table 5.11: Table with the averages, maximums and minimums of gyroscope test on PSoC platform in the different positions tested.

After analysing the output data from the module, it was produced tables with only the averages, the maximums and minimums of the values.

On the accelerometer test, both testing platforms produced similar results (tables 5.6 and 5.9).

On the magnetometer test, also the results obtained were practically similar on the Arduino testing platform and on the PSoC testing platform (tables 5.7 and 5.10).

On the Gyroscope test, although they have similar results on both testing plaforms, it is a sensor that changes his data very quickly (tables 5.8 and 5.11).

It's important to mention, that the magnetometer test was done without any magnet around, that could affect the sensor readings.

5.1.5 RTC

The RTC test was made to observe if the clock, even with rest of electronics shut down, would keep working and register correct data.

Although it's not a testing with days or moths to test effectively the accuracy of the module, it can represent a approximation of a good functionality.

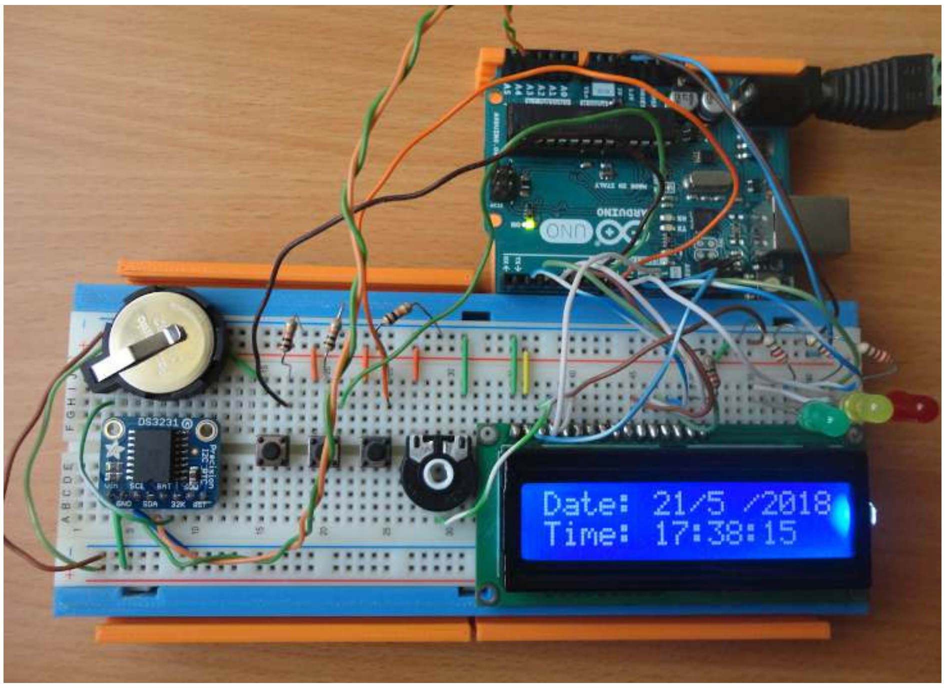


Figure 5.23: RTC testing on Arduino.

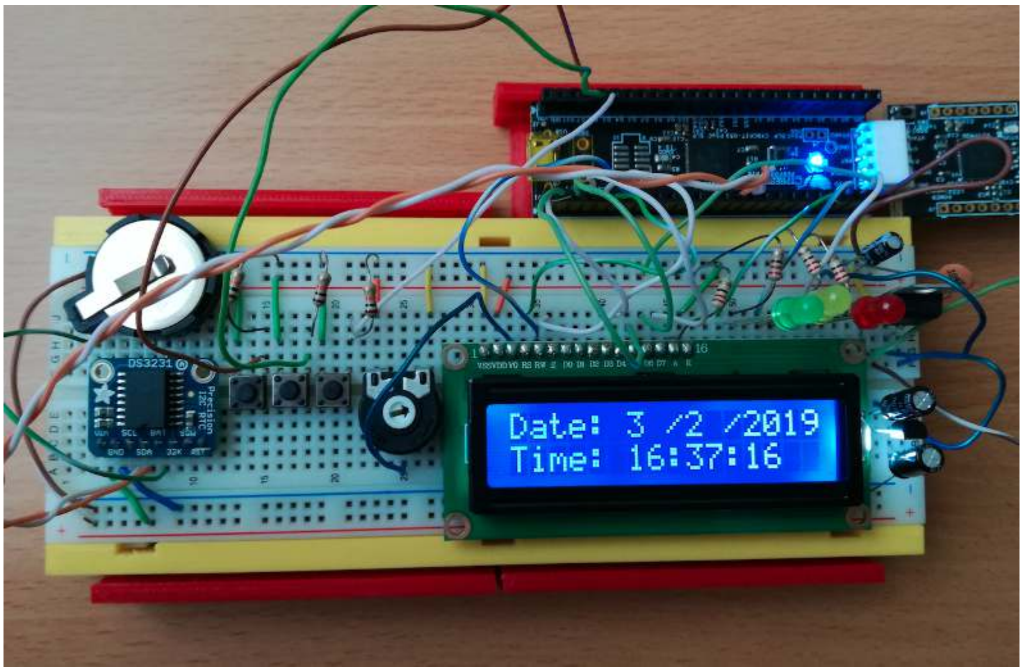


Figure 5.24: RTC testing on PSoC.

After the test were finished on both platforms, it can be concluded that the module works very accurate, compared to chronometer of a smartphone, and even when without the main power supply, powered only by the 3 V coin battery.

5.2 Other Payload Tests

5.2.1 PSOC and Arduino Communication

The communication between PSoC and Arduino through I^2C allowed to establish a communication channel, so the data could be transferred from PSoC to Arduino.

The test demonstrated that the data was always transferred to Arduino by the PSoC (figure 5.26). Although the data is not transferred every second, the PSoC performs data change every second.

It was observed that this test can failed if one of the controllers is reseted in the middle of information transmission, as this test was designed with transmissions flags to ensure the correct information was sent or received.

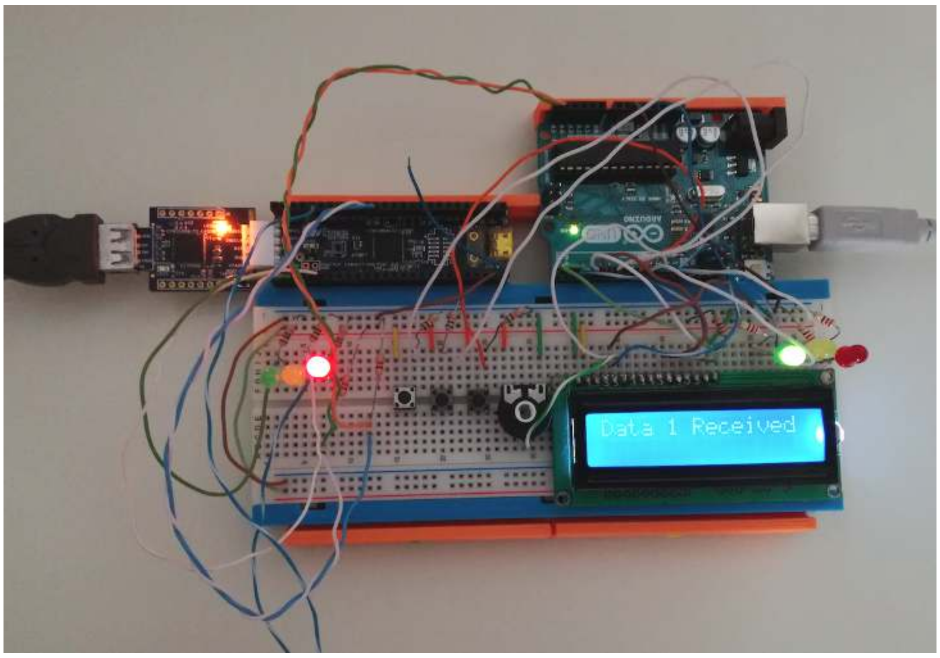
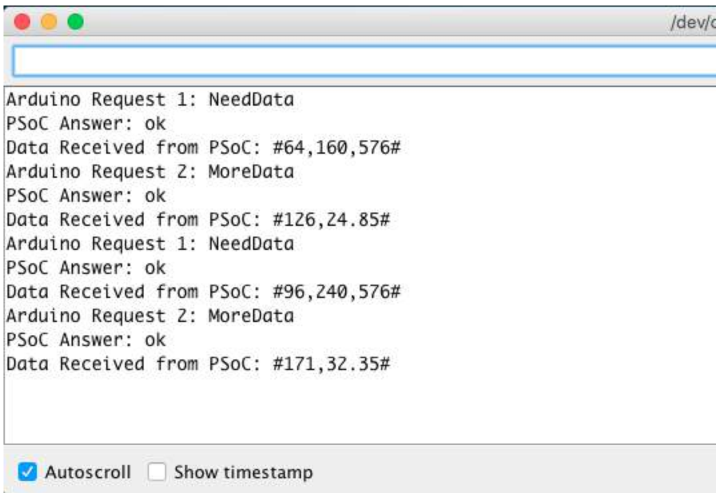


Figure 5.25: I^{2}C communication testing between Arduino and PSoC.



```
Arduino Request 1: NeedData
PSoC Answer: ok
Data Received from PSoC: #64,160,576#
Arduino Request 2: MoreData
PSoC Answer: ok
Data Received from PSoC: #126,24.85#
Arduino Request 1: NeedData
PSoC Answer: ok
Data Received from PSoC: #96,240,576#
Arduino Request 2: MoreData
PSoC Answer: ok
Data Received from PSoC: #171,32.35#
```

Figure 5.26: I^{2}C received data on Arduino in communication test between Arduino and PSoC.

5.2.2 Voltage Step-up and Battery Charger

The voltage step-up and battery charger test provided the useful information of when it is necessary to charge the back battery or if it is safe to charge that battery.

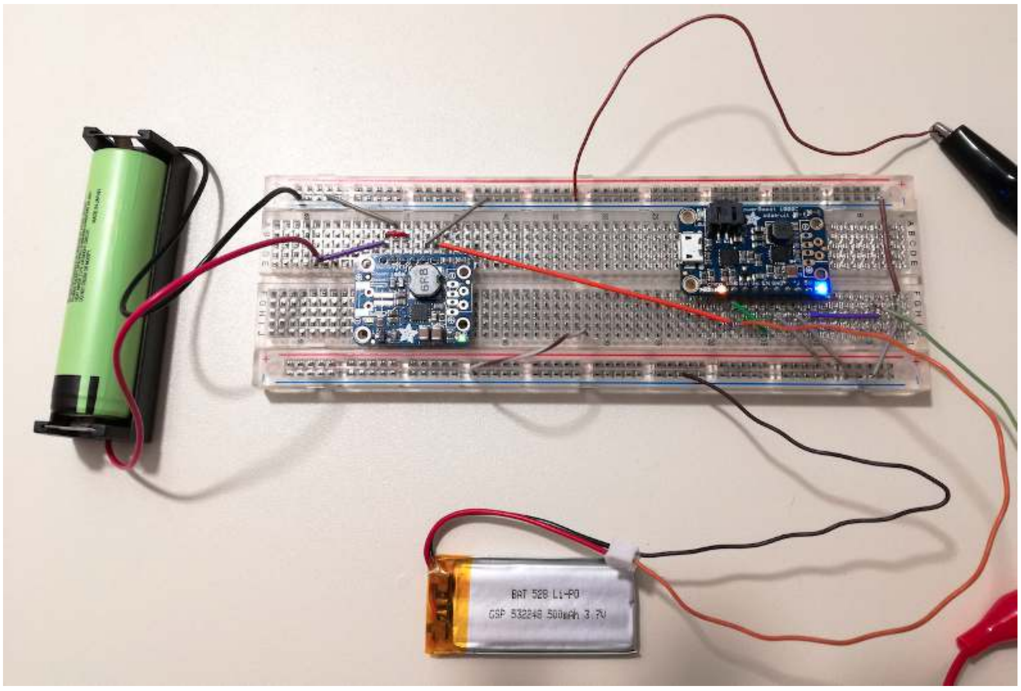
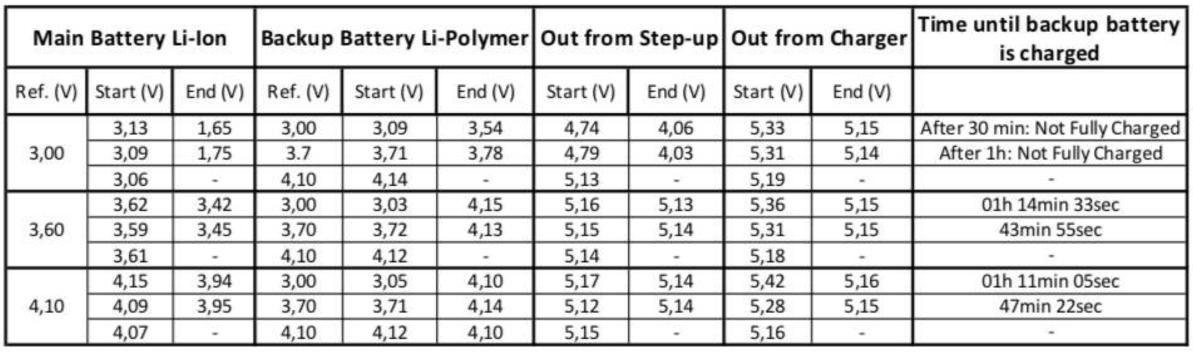


Figure 5.27: Voltage step-up and Battery Charger testing.



Main Battery Li-Ion			Backup Battery Li-Polymer			Out from Step-up		Out from Charger		Time until backup battery is charged
Ref. (V)	Start (V)	End (V)	Ref. (V)	Start (V)	End (V)	Start (V)	End (V)	Start (V)	End (V)	
3,00	3,13	1,65	3,00	3,09	3,54	4,74	4,06	5,33	5,15	After 30 min: Not Fully Charged
	3,09	1,75	3.7	3,71	3,78	4,79	4,03	5,31	5,14	After 1h: Not Fully Charged
	3,06	-	4,10	4,14	-	5,13	-	5,19	-	-
3,60	3,62	3,42	3,00	3,03	4,15	5,16	5,13	5,36	5,15	01h 14min 33sec
	3,59	3,45	3,70	3,72	4,13	5,15	5,14	5,31	5,15	43min 55sec
	3,61	-	4,10	4,12	-	5,14	-	5,18	-	-
4,10	4,15	3,94	3,00	3,05	4,10	5,17	5,14	5,42	5,16	01h 11min 05sec
	4,09	3,95	3,70	3,71	4,14	5,12	5,14	5,28	5,15	47min 22sec
	4,07	-	4,10	4,12	4,10	5,15	-	5,16	-	-

Table 5.12: Chart of comparation between temperature measurements on arduino platform with rtd using quations and using mapping function.

Through the table 5.12 is possible to observe that when the backup is charged, there is no charging to occur, so there is only charging movement when it's needed.

The output voltage from the step-up is almost the same for all cases (except when the main voltage is 3 V). The output of the charger module that also have a step-up built-in, maintains practically the same voltage on all cases.

The situations where it is mentioned that the backup battery was not fully charged means that the main Li-Ion battery was not delivering energy due to low battery voltage produced by the high voltage drop originated by the necessity of transferring energy to the backup battery. The low battery voltage from the main battery (reference of 3 V) means that the battery is fully discharged and can't take the operation of charging another battery. Also when the battery is in this case, any voltage drop could provoke a cut-off of the battery itself.

5.3 Testing Payload Board Results

The payload board was tested to accomplish a final stage test where is encompassed all the modules and sensors tested previously.

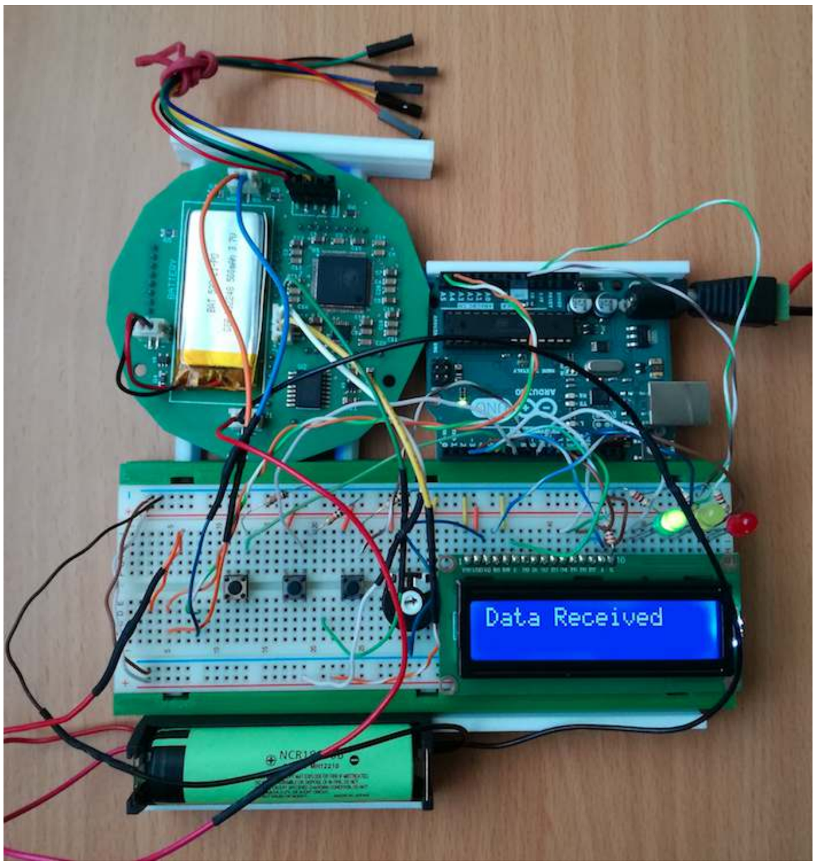
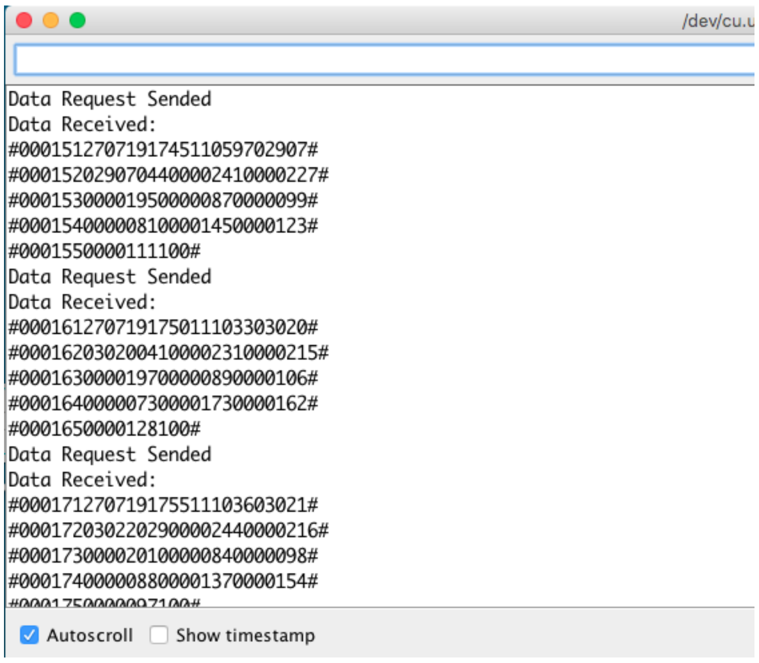


Figure 5.28: CTSAT-1 payload board test.

In figure 5.29 is represented one batch of data sent by the payload board to Arduino.

The data shown, it's not Arduino's job to decode the information, as the Arduino will re-transmit the information data to the Earth's ground station, where will be later deciphered.



```
Data Request Sended
Data Received:
#00015127071917451105970290 7#
```

Figure 5.29: CTSAT-1 sent data and received by Arduino.

In the table 5.13 is shown the power consumption measurements during various situations.

The current measurement was higher than expected, but it's because some modules have integrated leds to show some signal states of that same module.

Averages (mA)			
Standby	Sensors / Modules Reading	Sending Data Through I2C	Receive Data Through I2C
51,07	53,23	53,73	52,17

Table 5.13: CTSAT-1 power consumption.

Chapter 6

Final considerations and Future Work.

6.1 Conclusions

This work consisted on a payload development for a picosatellite of tubesat type. That consisted in choose components, design PCB layout, test the components, soldering, testing and developing the software.

The mock-up of the picosatellite made with 3D printing technologies proven very useful, as to understand its structure and as to provide a base for the payload PCB layout. The components chosen provided the measurements requirements for this project.

The payload system designed architecture is responsible for the management of data reading of the sensors and modules, the communications lines priorities between PSoC, Arduino and modules, the timing events and the errors handling.

Under sensor's individuals tests with Arduino, every sensor worked as aspected with big help of already made libraries for the modules used in the payload.

Changing the Arduino for the PSoC revealed more work to be done than initially expected, regarding the communications with modules and sensors. Components datasheets usually details how can data be handled but sometimes, the information is not very explicit. This bunch of tests also ended to work as aspected.

For the payload complete PCB tests, the system behaved normally at the registering data and sent it to the controller Arduino.

6.2 Future Work

This project is not ready for a space flight as there are still many things to do. Until the moment of flight, is necessary to ensure that everything works as expected.

From the payload point of view, as the picosatellite is powered by batteries, it is necessary to built-in a power sleep mode into the PSoC to minimise the power consumption when there aren't communications and when there aren't any measurements being received from the sensors. Also to minimise the payload's consumption it is necessary to remove the integrated leds present in some modules. As there are no physical access in space, a better error handling could be made to ensure a superior approach with more errors being detected and some being corrected. It is necessary to proceed to the activation of the Flash memory implementation present in the payload. Moreover, it is necessary to prevent infinite cycles can occur during operation.

In order to have a better management of available resources, there may be a migration of other components present on the other picosatelite layers to this payload layer.

In the other stages, it is necessary to build them and test, with special attention to radio level. A suitable Software Defined Radio must be chosen to provide a simple and reliable communications between the picostallite and the Earth station.

In the end, a resistance and environmental conditions tests must be done to ensure a proper working the in the LEO region.

Bibliography

[1] *AAUSAT - Aalborg University*. URL: http://www.space.aau.dk/cubesat/ (visited on 04/12/2016).

[2] *About QuakeSat*. URL: https://www.nasa.gov/mission_pages/station/research/experiments/1326.html (visited on 07/01/2017).

[3] *About QuakeSat*. URL: https://www.quakefinder.com/science/about-quakesat/ (visited on 07/01/2017).

[4] A. Mehrparvar et al. *Cubesat Design Specification (CDS) rev.13*. California Polytechnic State University, 2014.

[5] C. Boshuizen et al. *Results from the Planet Labs Flock Constellation*. 2014.

[6] C. Kitts et al. *Initial Flight Results from the PharmaSat Biological Microsatellite Mission*. 2009.

[7] H. Klinkrad et al. *Update of the ESA Space Debris Mitigation Handbook*. 2002.

[8] L. Alminde et al. *Educational Value and Lessons Learned from the AAU-CubeSat Project*. 2003.

[9] M. Long et al. *A CubeSat Derived Design for a Unique Academic Research Mission in Earthquake Signature Detection*. 16th Annual/USU Conference on Small Satellites, 2002.

[10] T. Gansmoe et al. *The CanSat Book*. NAROM, 2015.

[11] T. Bleier et al. *QuakeSat Lessons Learned: Notes from the Development of a Triple CubeSat*. QuakeFinder, LLC, 2004.

[12] *Ames Research Center 20-G Centrifuge*. URL: https://www.nasa.gov/ames/research/space-biosciences/20-g-centrifuge (visited on 10/02/2017).

[13] S. Antunes. *DIY Satellite Platforms: Building a Space-Ready Base Picosatellite for Any Mission*. O'Reilly Media, Inc, 2012.

[14] S. Antunes. *Surviving Orbit the DIY Way: Testing the Limits Your Satellite Can and Must Match*. O'Reilly Media, Inc, 2012.

[15] S. Antunes. *DIY Instruments for Amateur Space: Inventing Utility for Your Spacecraft Once it Achieves Orbit*. O'Reilly Media, Inc, 2013.

[16] *Application Note: Ultraviolet Sensor IC with Voltage Output*. Lapis Semiconductor Co., Ltd.

[17] *Atomic Oxygen - NASA's Glenn Research Center*. URL: https://www.nasa.gov/topics/technology/features/atomic_oxygen.html (visited on 03/02/2017).

[18] *Aurora picture by ISS*. URL: https://blogs.nasa.gov/spacestation/2017/02/03/astronauts-study-new-skinsuit-and-new-lights/blog_iss049e024763/ (visited on 07/02/2017).

[19] *DS3231 Datasheet: Extremely Accurate I2C-Integrated RTC/TCXO/Crystal*. Maxim Integrated Products, Inc., Mar. 2015.

[20] *Engineering Test Facilities Guide*. National Aeronautics and Space Administration.

[21] C. A. Esionwu Jnr. *Applications of CubeSats*. 2014.

[22] C. A. Esionwu Jnr. *List of CubeSat Missions*. 2014.

[23] P. Fortescue, G. Swinerd, and J. Stark. *Spacecraft Systems Engineering*. John Wiley & Sons, Ltd, 2011.

[24] *Go into a NASA Clean Room Daily with the Webb Telescope*. URL: http://www.esa.int/Education/2016_European_CanSat_competition_winners (visited on 10/01/2017).

[25] *Go into a NASA Clean Room Daily with the Webb Telescope*. URL: https://www.nasa.gov/centers/goddard/news/features/2010/webb-cam.html (visited on 10/02/2017).

[26] J. R. Heirtzler. *The Future of the South Atlantic Anomaly And Implications for Radiation Damage in Space*. Journal of Atmospheric and Solar-Terrestrial Physics, 2002.

[27] *Hydraulic Shaker - ESTEC Test Centre*. URL: http://www.esa.int/Our_Activities/Space_Engineering_Technology/Test_centre/Hydraulic_shaker (visited on 09/02/2017).

[28] *Hypergravity Research*. URL: http://www.esa.int/Our_Activities/Human_Spaceflight/Research/Hypergravity (visited on 10/02/2017).

[29] *Interorbital TubeSat Kits On Orbit and Functioning*. URL: http://www.interorbital.com/interorbital_06222015_021.htm (visited on 05/02/2017).

[30] *IOS Orbital Launch Manifest*. URL: http://www.interorbital.com/interorbital_06222015_014.htm (visited on 08/12/2016).

[31] C. Kakoyiannis and P. Constantinou. *Electrically Small Microstrip Antennas Targeting Miniaturized Satellites : the CubeSat Paradigm*. 2011.

[32] *Large European Acoustic Facility (LEAF)*. URL: http://www.esa.int/Our_Activities/Space_Engineering_Technology/Test_centre/Large_European_Acoustic_Facility_LEAF (visited on 09/02/2017).

[33] J. C. Liou and N. L. Johnson. *Risks in Space from Orbiting Debris*. 2006.

[34] C. A. P. Loura. *Sistema de Aquisição de Sinal para um satélite de baixo custo do tipo Tubesat*. 2016.

[35] *LSM9DS0 Datasheet: iNEMO inertial module: 3D Accelerometer, 3D Gyroscope, 3D Magnetometer*. STMicroeletronics, 2013.

[36] A. Marinan and K. Cahoy. *From CubeSats to Constellations: Systems Design and Performance Analysis*. Massachusetts Institute of Technology, 2013.

[37] *MAX3372E-MAX3379E/MAX3390E-MAX3393E Datasheet: 15kV ESD-Protected, 1uA, 16Mbps, Dual/Quad Low-Voltage Level Translator in UCSP*. Maxim Integrated Products, Inc., 2013.

[38] *MCP7381 Datasheet: Stand-Alone System Load Sharing and Li-Ion/Li-Polymer Battery Charge Management Controller*. Microchip Technology Inc., 2009.

[39] *MCP9808 Datasheet: 0.5ºC Maximum Accuracy Digital Temperature Sensor*. Microchip Technology Inc., 2011.

[40] *MIC5225 Datasheet: Ultra-Low Quiescent Current 150mA uCap Low Dropout Regulator*. Micrel Inc., 2008.

[41] *Mission Objectives of QB50 Project*. URL: https://www.qb50.eu/index.php/project-description-obj/mission-objectives (visited on 12/01/2017).

[42] *ML8511 Datasheet: UV Sensor with Voltage Output*. Lapis Semiconductor Co., Ltd., 2013.

[43] *NAND vs. NOR Flash Memory: Tecnology Overview*. Toshiba Corporation, Apr. 2006.

[44] *NASA - Pharmasat*. URL: https://www.nasa.gov/mission_pages/smallsats/pharmasat/main/index.html (visited on 07/01/2017).

[45] *NASA Facts : PharmaSat Nanosatellite*. NASA Ames Research Center.

[46] *NASA Restarts Rigorous Vibration Testing on the James Webb Space Telescope*. URL: https://www.nasa.gov/feature/goddard/2017/nasa-restarts-rigorous-vibration-testing-on-the-james-webb-space-telescope (visited on 09/02/2017).

[47] *Phenix Thermal Vacuum Chamber*. URL: http://www.esa.int/Our_Activities/Space_Engineering_Technology/Test_centre/Phenix_Thermal_Vacuum_Chamber (visited on 09/02/2017).

[48] *Planet Labs Gallery*. URL: https://www.planet.com/gallery/ (visited on 07/01/2017).

[49] *PSoC 5LP: CY858LP Family Datasheet*. Cypress Semiconductor Corporation, 2012.

[50] *PTS AT Datasheet: Platinum SMD Flat Chip Temperature Sensor*. Vishay InterTechnology, Inc, Dec. 2015.

[51] *Reverberant Acoustic Test Facility (RATF) at NASA Glenn Research Center*. URL: https://facilities.grc.nasa.gov/spf/ (visited on 09/02/2017).

[52] *Space Debris in LEO*. URL: https://orbitaldebris.jsc.nasa.gov/photo-gallery.html (visited on 02/02/2017).

[53] *STMPS2141, STMPS2151, STMPS2161, STMPS2171 Datasheet: Enhanced Single Channel Power Switches*. STMicroeletronics, Jan. 2013.

[54] *TC58CYG0S3HxAIx Datasheet: Toshiba Serial Interface NAND Technical Data Sheet*. Toshiba Corporation, Nov. 2016.

[55] *Tech Note: Callendar-Van Dusen Equation and RTD Temperature Sensors*. ILX Lightwave.

[56] *TPS61030, TPS61031, TPS61032 Datasheet: 96% Efficient Synchronous Boost Converter with 4A Switch*. Texas Instruments Incorporated, Jan. 2012.

[57] *TPS61090, TPS61091, TPS61092 Datasheet: Synchronous Boost Converter with 2A Switch*. Texas Instruments Incorporated, 2004.

[58] *TubeSat Personal Satellite Kit*. URL: http://www.interorbital.com/interorbital_06222015_002.htm (visited on 09/12/2016).

[59] *TubeSat Personal Satellite Kit - Assembly Guide*. Interorbital Systems, 2012.

[60] *What is a CanSat?* URL: http://www.esa.int/Education/CanSat/What_is_a_CanSat (visited on 10/01/2017).

Annex I

CTSAT-1 Structure - Technical Files

I.1 Mock-up PCBs

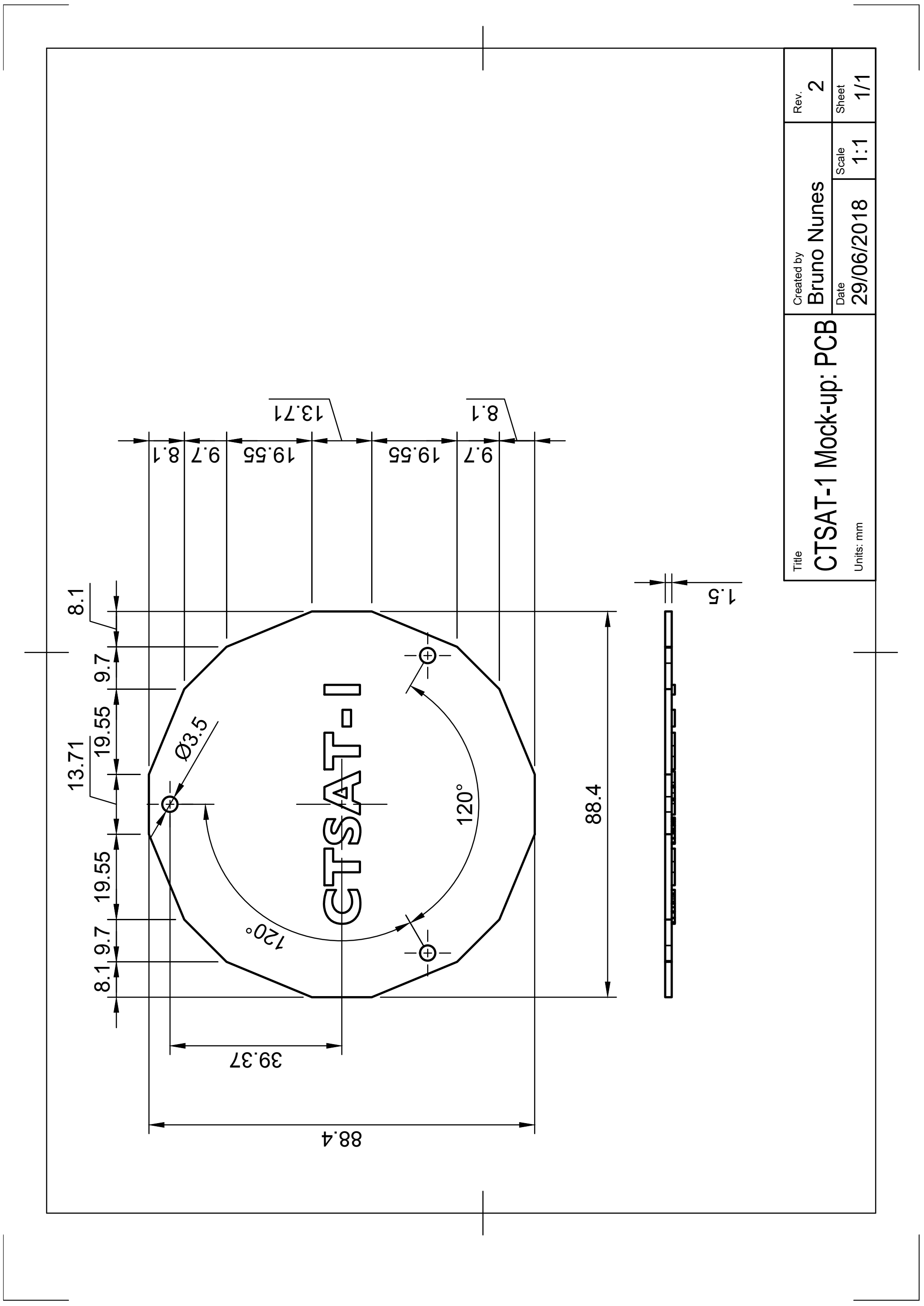


Figure I.1: CTSAT-1 PCBs dimensions used in the mock-up.

I.2 Mock-up Solar Panels PCB

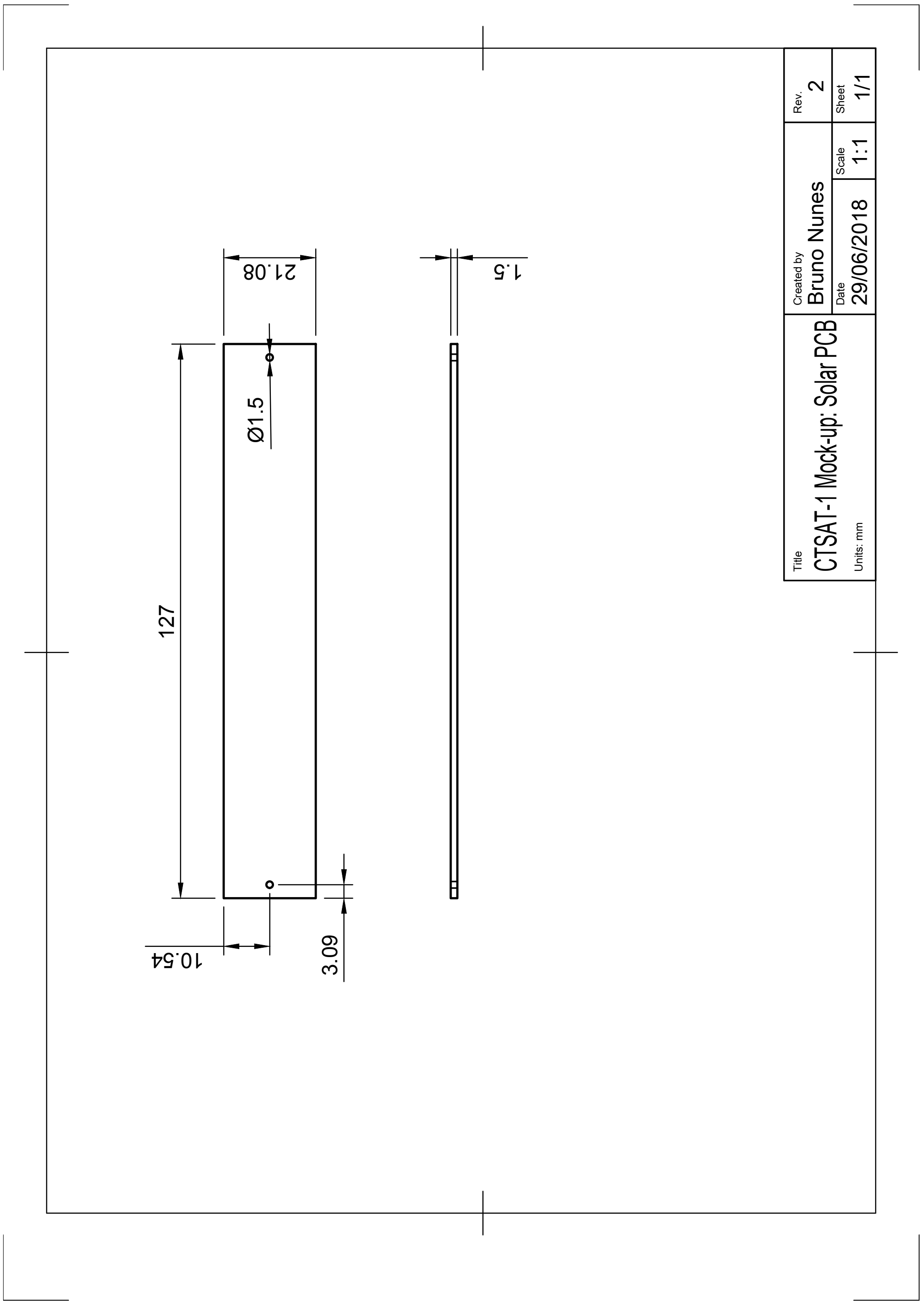


Figure I.2: CTSAT-1 solar PCB dimensions used in the mock-up.

I.3 Mock-up Aluminium Strips

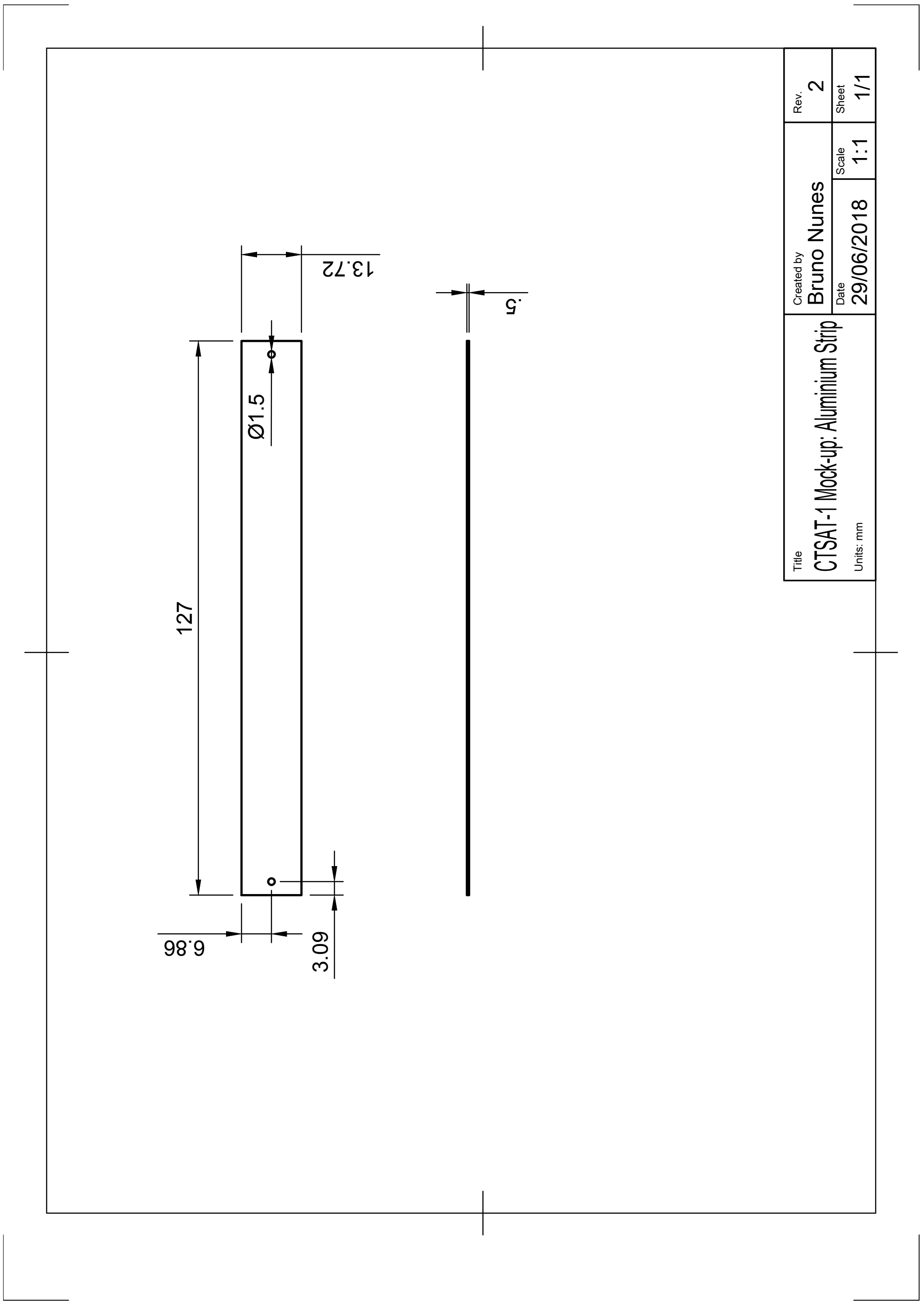


Figure I.3: CTSAT-1 aluminium strip dimensions used in the mock-up.

I.4 Mock-up Rings

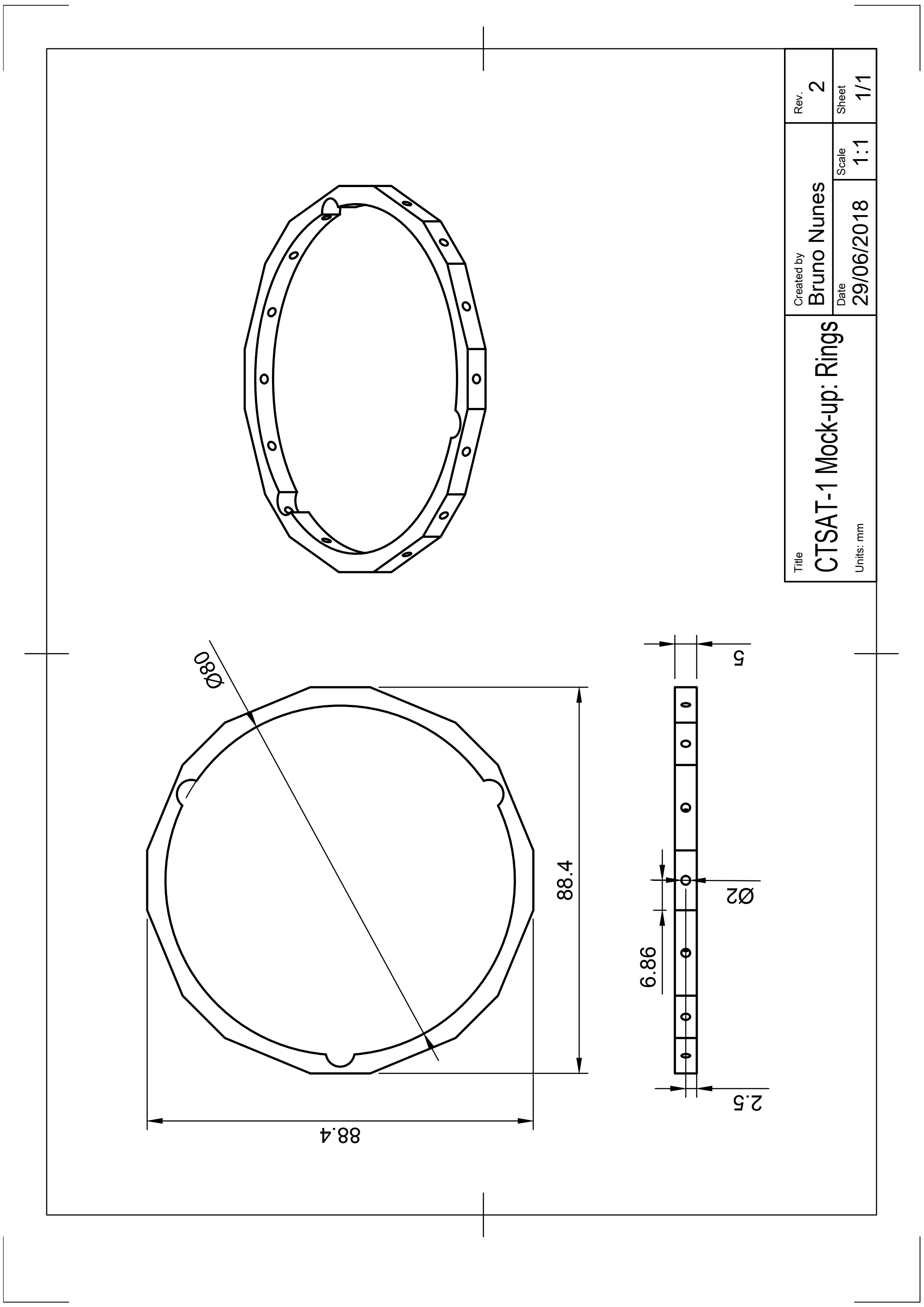


Figure I.4: CTSAT-1 rings dimensions used in the mock-up.

I.5 Rings

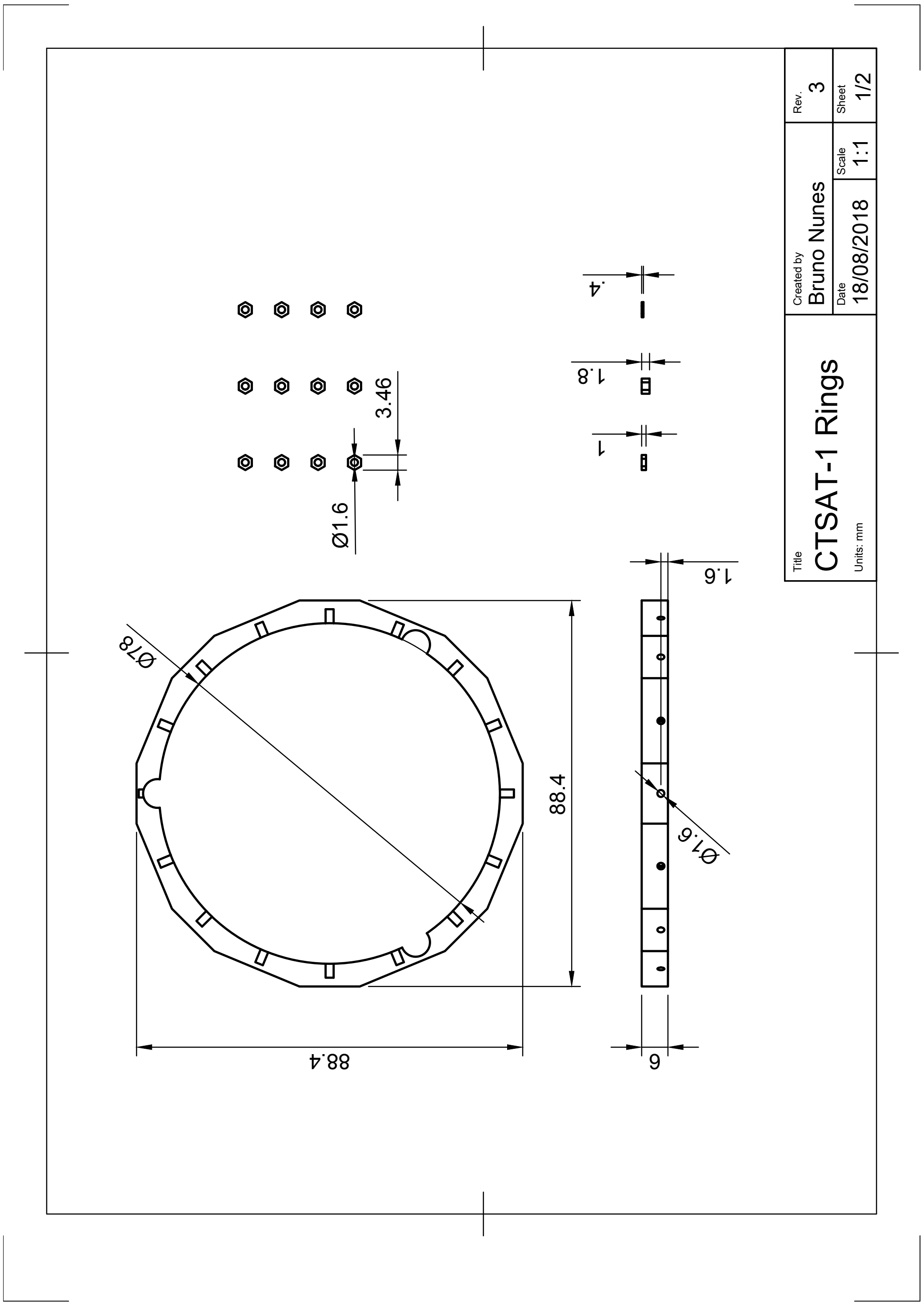


Figure I.5: CTSAT-1 rings dimensions.

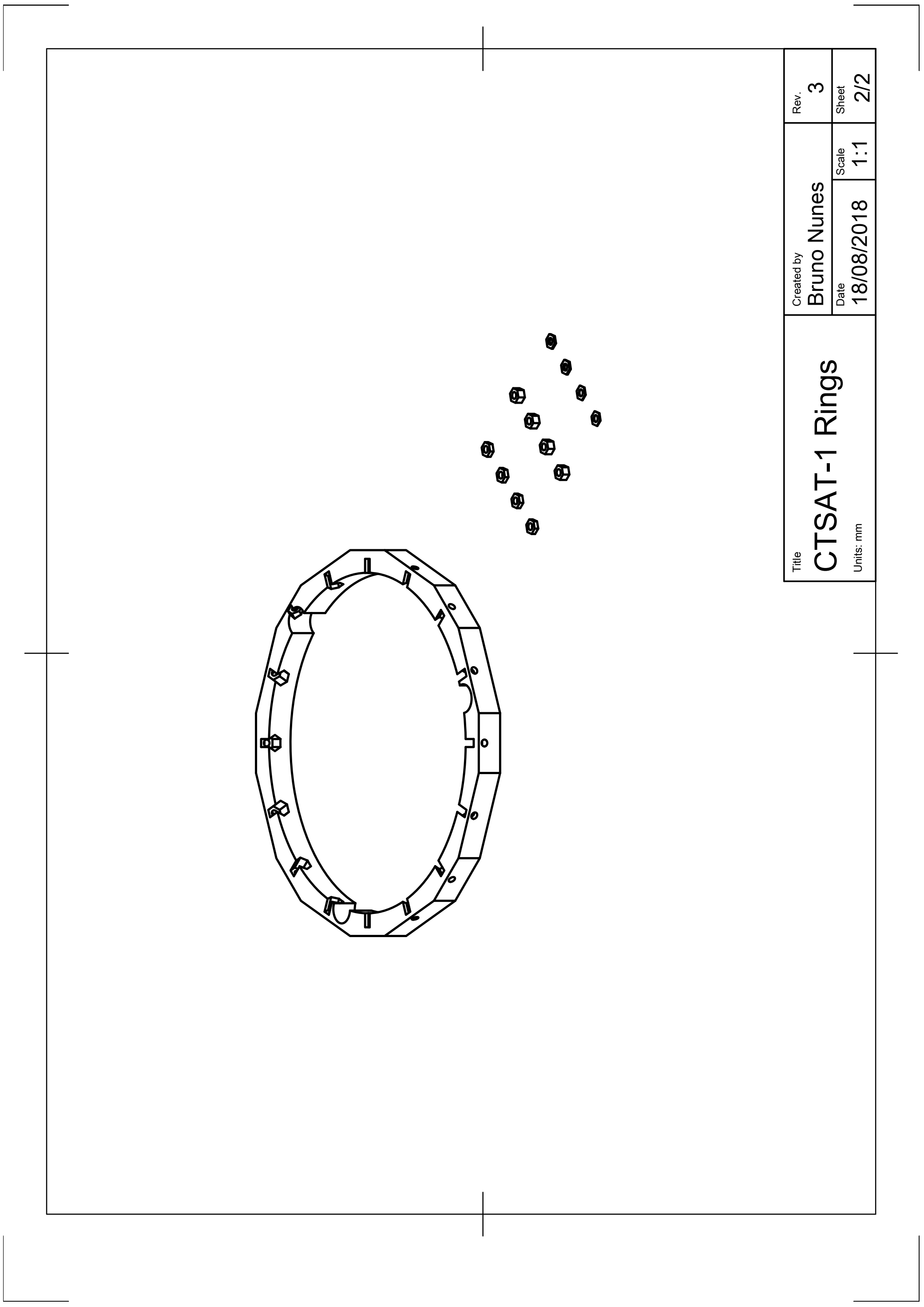


Figure I.6: CTSAT-1 rings - orthogonal projection.

Annex II

CTSAT-1 PCBs - Technical Files

II.1 Antenna

II.1.1 Schematic

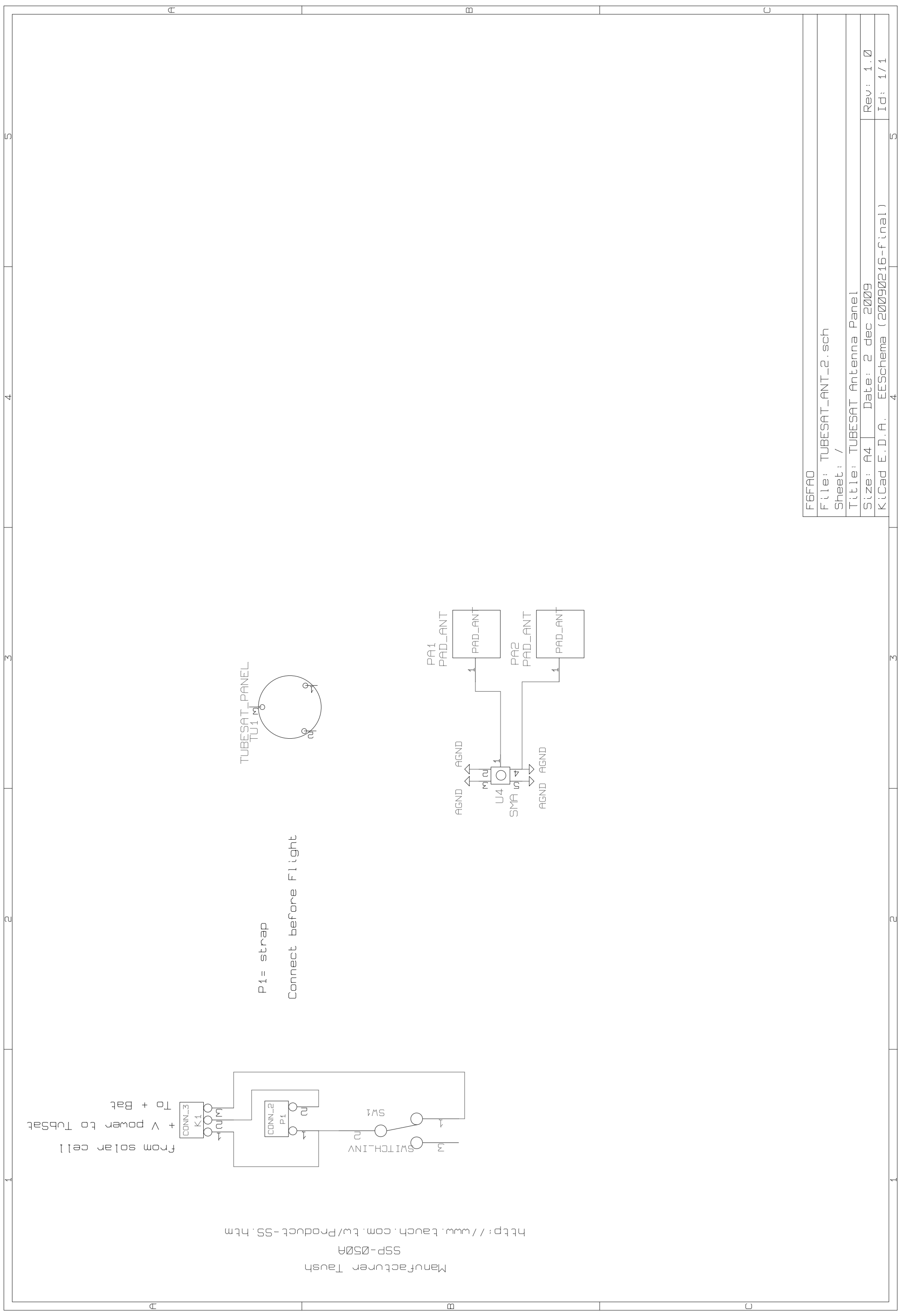


Figure II.1: Antenna schematic.

II.1.2 PCB Layout

II.1.2.1 Top Layer

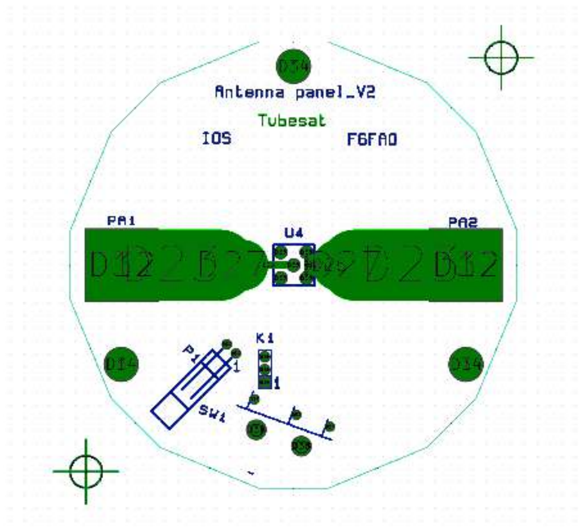


Figure II.2: Antenna PCB top layout.

II.1.2.2 Bottom Layer

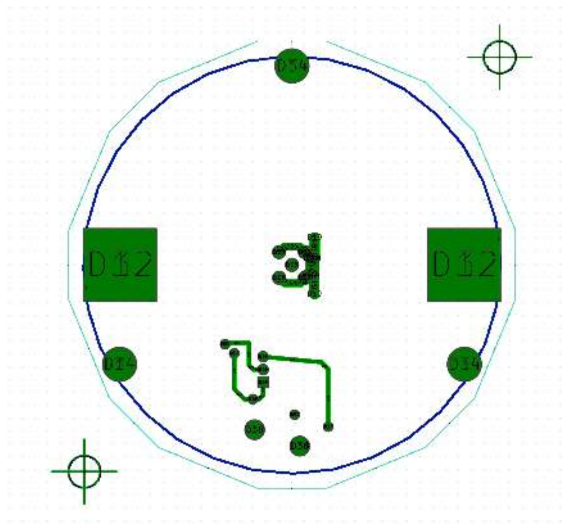
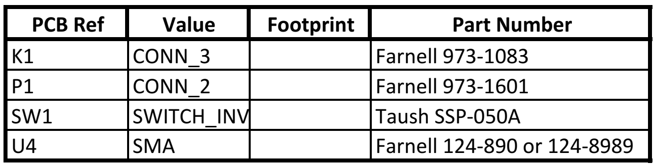


Figure II.3: Antenna PCB bottom layout.

II.1.3 List of Materials



PCB Ref	Value	Footprint	Part Number
K1	CONN_3		Farnell 973-1083
P1	CONN_2		Farnell 973-1601
SW1	SWITCH_INV		Taush SSP-050A
U4	SMA		Farnell 124-890 or 124-8989

Table II.1: List of materials for the antenna PCB.

II.2 Power Management

II.2.1 Schematic

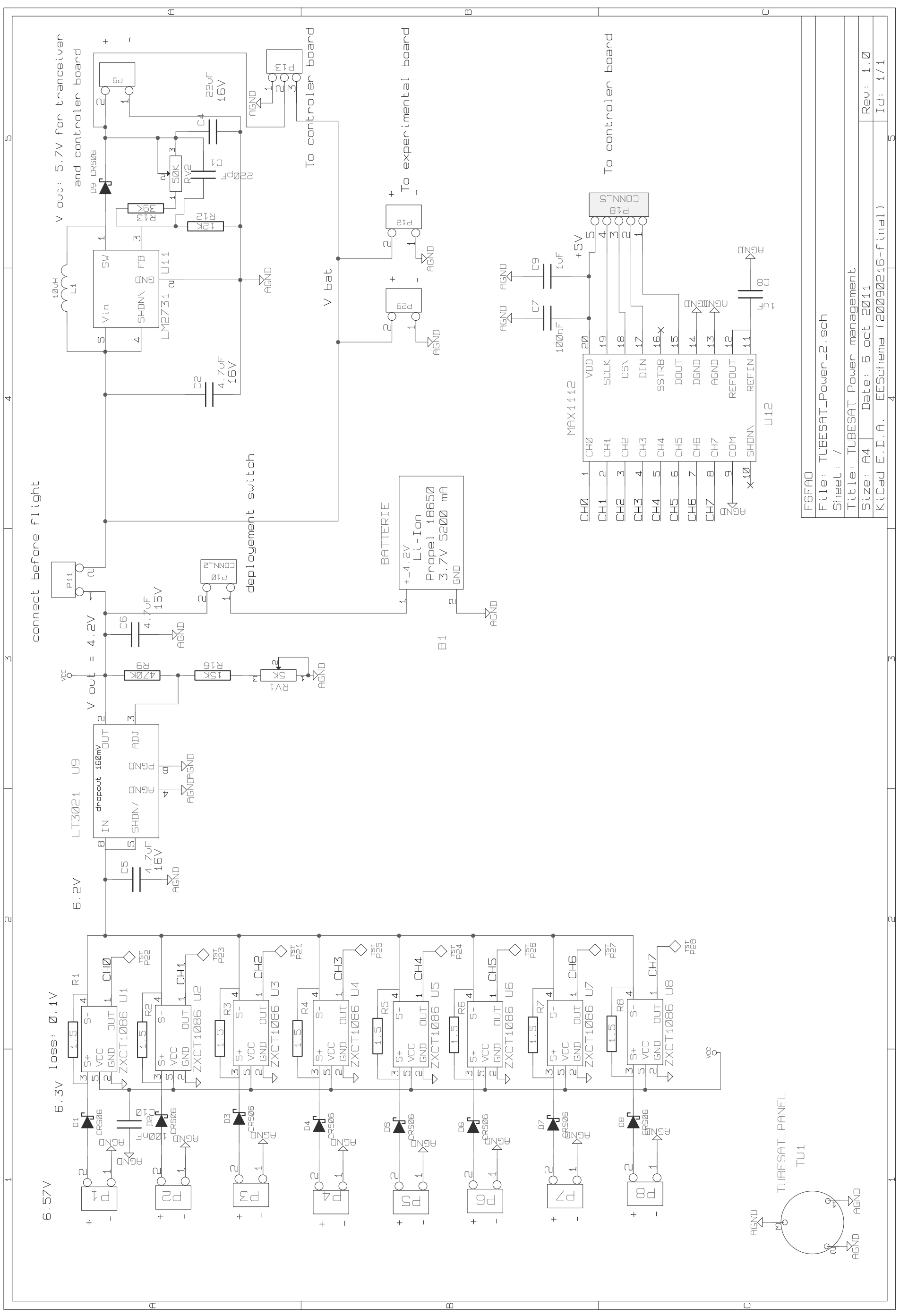


Figure II.4: Power management schematic.

II.2.2 PCB Layout

II.2.2.1 Top Layer

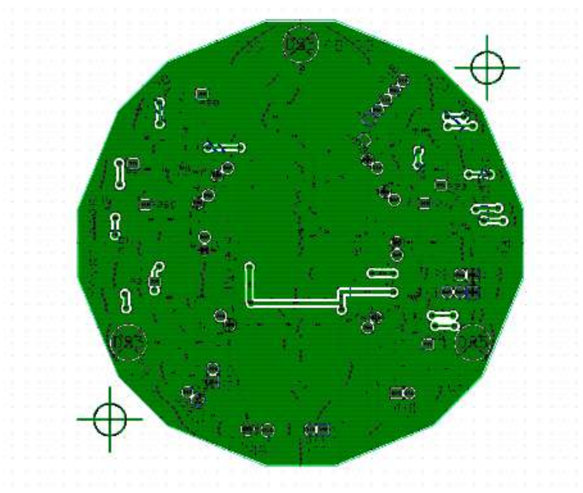


Figure II.5: Power management PCB top layout.

II.2.2.2 Bottom Layer

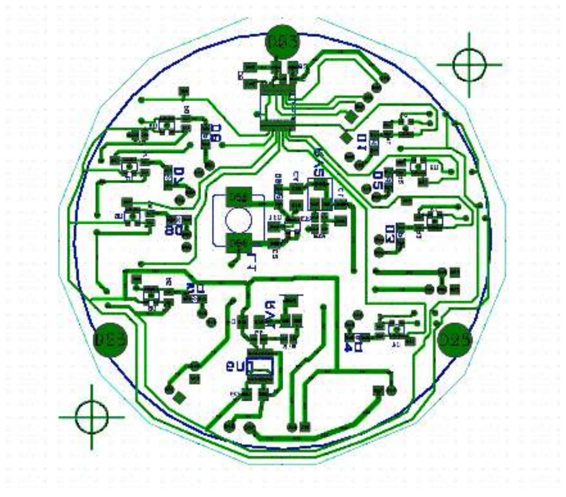


Figure II.6: Power management PCB bottom layout.

II.2.3 List of Materials

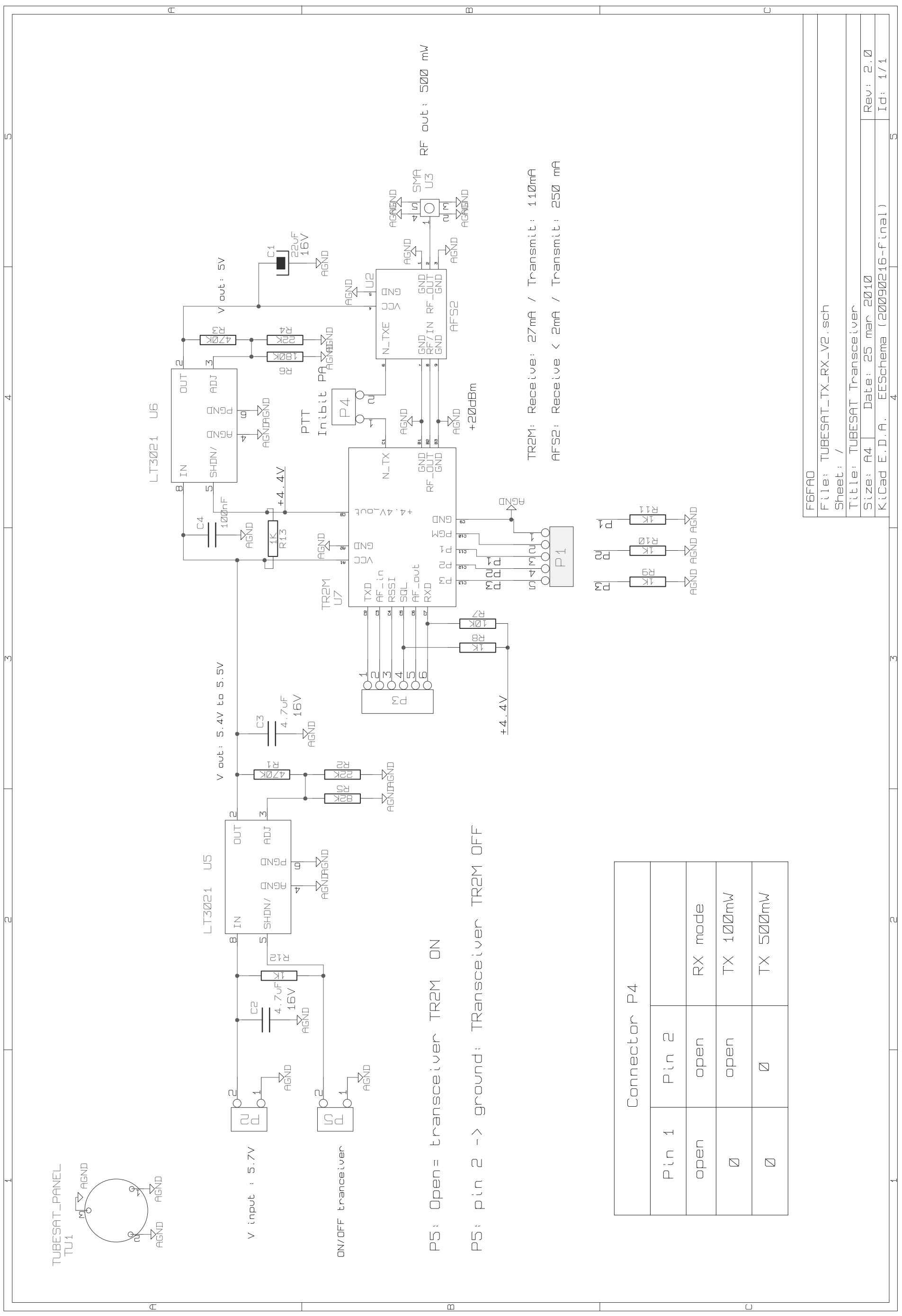
PCB Ref	Value	Footprint	Observations	Part Number
B1	BATTERIE			LI-Ion Propel 3.7V 2700mA
C1	220pF	1206		Farnell 940-6468
C2	4.7uF	1206	16V	Farnell 922-7946
C4	22uF	1206	16V	Farnell 952-7800
C5	4.7uF	1206	16V	Farnell 922-7946
C6	4.7uF	1206	16V	Farnell 922-7946
C7	100nF	SM0805	50V	Farnell 940-6387
C8	1uF	1206	25V	Farnell 922-7873
C9	1uF	1206	25V	Farnell 922-7873
C10	100nF	SM0805		Farnell 940-6387
D1	CRS06	SOD-323		Farnell 130-0795
D2	CRS06	SOD-323		Farnel 130-0795
D3	CRS06	SOD-323		Farnel 130-0795
D4	CRS06	SOD-323		Farnel 130-0795
D5	CRS06			Farnel 130-0795
D6	CRS06	SOD-323		Farnel 130-0795
D7	CRS06	SOD-323		Farnel 130-0795
D8	CRS06	SOD-323		Farnel 130-0795
D9	CRS06	SOD-323		Farnel 130-0795
L1	10uH		10x10 shielded E	Farnell 742-9444
P1	CONN_2			Farnell 973-1601
P2	CONN_2			Farnell 973-1601
P3	CONN_2			Farnell 973-1601
P4	CONN_2			Farnell 973-1601
P5	CONN_2			Farnell 973-1601
P6	CONN_2			Farnell 973-1601
P7	CONN_2			Farnell 973-1601
P8	CONN_2			Farnell 973-1601
P9	CONN_2			Farnell 973-1075
P10	CONN_2			Farnell 973-1075
P11	CONN_2			Farnell 9731199
P12	CONN_2			Farnell 973-1075
P13	CONN_3			Farnell 973-1083
P18	CONN_5			Farnell 973-1105
P29	CONN_2			Farnell 973-1075
R1	1,5	SM0805		Farnell 933-3959
R2	1,5	SM0805		Farnell 933-3959
R3	1,5	SM0805		Farnell 933-3959
R4	1,5	SM0805		Farnell 933-3959
R5	1,5	SM0805		Farnell 933-3959
R6	1,5	SM0805		Farnell 933-3959
R7	1,5	SM0805		Farnell 933-3959
R8	1,5	SM0805		Farnell 933-3959
R9	470K	SM0805		Farnell 933-4602
R12	12K	SM0805		Farnell 933-3835
R13	39K	SM0805		Farnell 933-4491
R16	15K	SM0805		Farnell 933-3932
RV1	5K	POT-H		Farnell 514-822
RV2	50K	POT-H		Farnell 514-858
U1	ZXCT1086	SOT23-5		Farnell 190-4029
U2	ZXCT1086	SOT23-5		Farnell 190-4029
U3	ZXCT1086	SOT23-5		Farnell 190-4029
U4	ZXCT1086	SOT23-5		Farnell 190-4029
U5	ZXCT1086	SOT23-5		Farnell 190-4029
U6	ZXCT1086	SOT23-5		Farnell 190-4029

U7	ZXCT1086	SOT23-5		Farnell 190-4029
U8	ZXCT1086	SOT23-5		Farnell 190-4029
U9	LT3021	SO8-narrow		Farnell 127-3637
U11	LM2731	SOT23-5		Farnell 818-1640
U12	MAX1112	SSOP-20maxim		Farnell 137-9848

Table II.2: List of materials for the power management PCB.

II.3 Transceiver

II.3.1 Schematic


F6FAO
File: TUBESAT_TX_RX_V2.sch
Sheet: /
Title: TUBESAT Transceiver
Size: A4 Date: 25 mar 2010
Rev: 2.0
KiCad E.D.A. EESchema (20090216-final)
Id: 1/1
RF out: 500 mW
TR2M: Receive: 27mA / Transmit: 110mA
AFS2: Receive < 2mA / Transmit: 250 mA
V out: 5V
V out: 5.4V to 5.5V
V input : 5.7V
ON/OFF tranceiver
+4.4V
+20dBm
PTT
Inibit PA
P5: Open= transceiver TR2M ON
P5: pin 2 -> ground: TRansceiver TR2M OFF
TUBESAT_PANEL
TU1

Connector P4

Pin 1	Pin 2	
open	open	RX mode
Ø	open	TX 100mW
Ø	Ø	TX 500mW

Figure II.7: Transceiver schematic.

II.3.2 PCB Layout

II.3.2.1 Top Layer

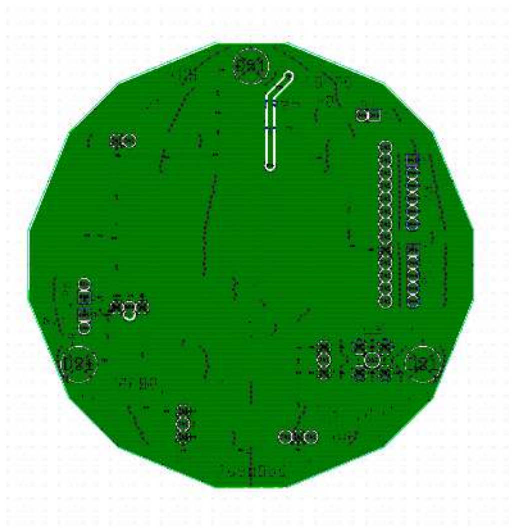


Figure II.8: Transceiver PCB top layout.

II.3.2.2 Bottom Layer

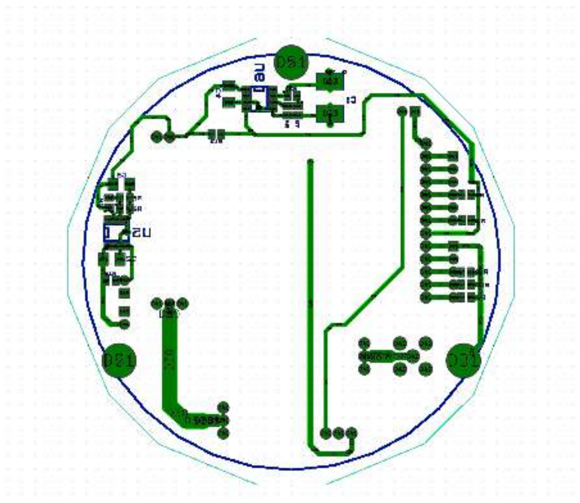


Figure II.9: Transceiver PCB bottom layout.

II.3.3 List of Materials

PCB Ref	Value	Footprint	Observations	Part Number
C1	22uF	1210	16V	Farnell 952-7800
C2	4.7uF	1206	16V	Farnell 922-7946
C3	4.7uF	1206	16V	Farnell 922-7946
C4	100nF	1206		Farnell 940-6557
P1	CONN_5			Farnell 973-1105
P2	CONN_2			Farnell 973-1075
P3	CONN_6			Farnell 973-1113
P4	CONN_2			Farnell 973-1075
P5	CONN_2			Farnell 973-1075
R1	470K	SM0805		Farnell
R2	22K	SM0805		Farnell
R3	470K	SM0805		Farnell
R4	22K	SM0805		Farnell
R5	82K	SM0805		To be ajusted to get 5.5V
R6	180K	SM0805		To be ajusted to get 5V
R7	10K	SM0805		Farnell
R8	1K	SM0805		Farnell
R9	1K	SM0805		Farnell
R10	1K	SM0805		Farnell
R11	1K	SM0805		Farnell
R12	1K	SM0805		Farnell
R13	1K	SM0805		Farnell
U2	AFS2			Radiometrix
U3	SMA			Farnell 116-9631
U5	LT3021			Farnell 127-3637
U6	LT3021			Farnell 127-3637
U7	TR2M			Radiometrix

Table II.3: List of materials for the transceiver PCB.

II.4 Microcontroller

II.4.1 Schematic

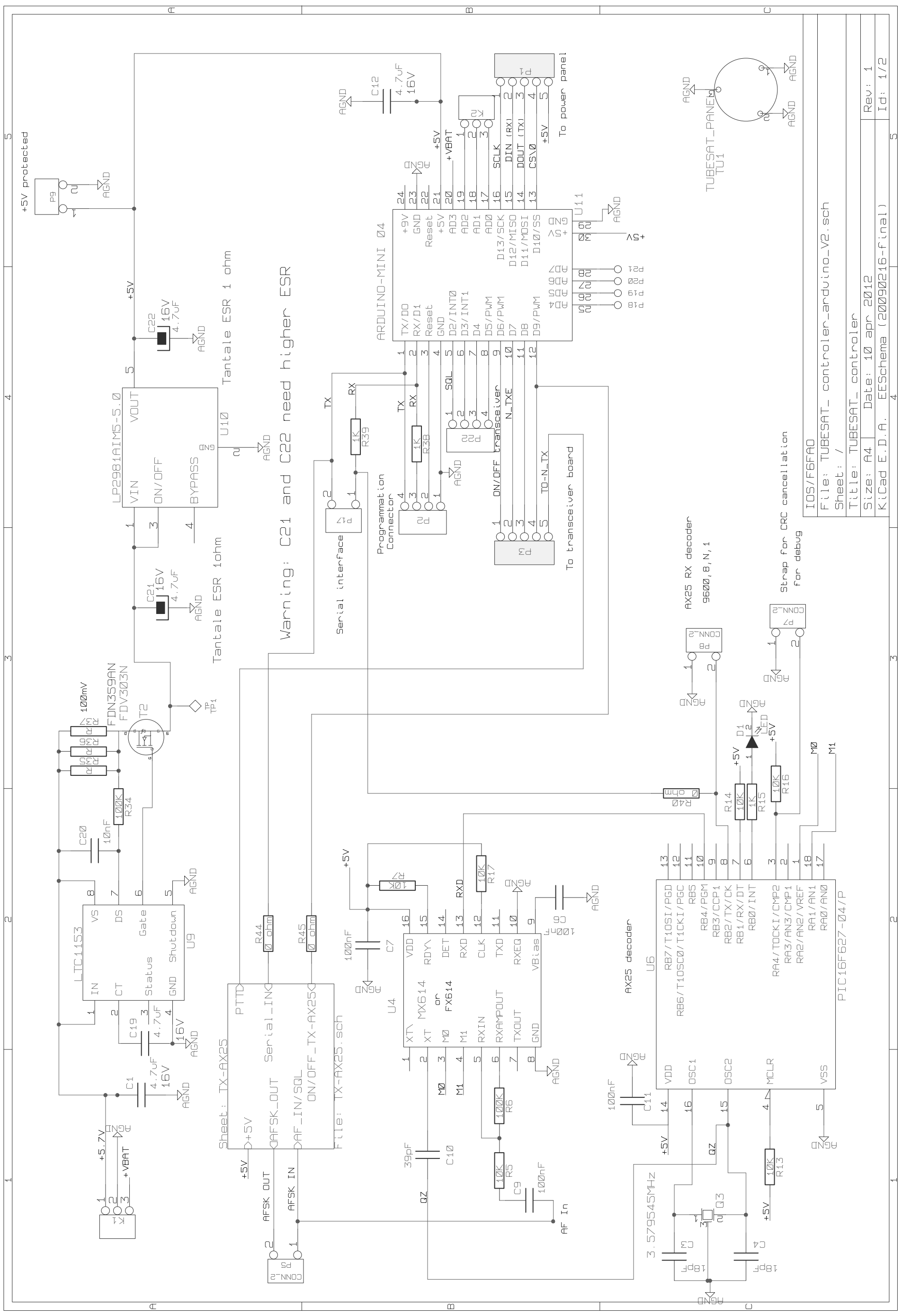


Figure II.10: Microcontroller schematic.

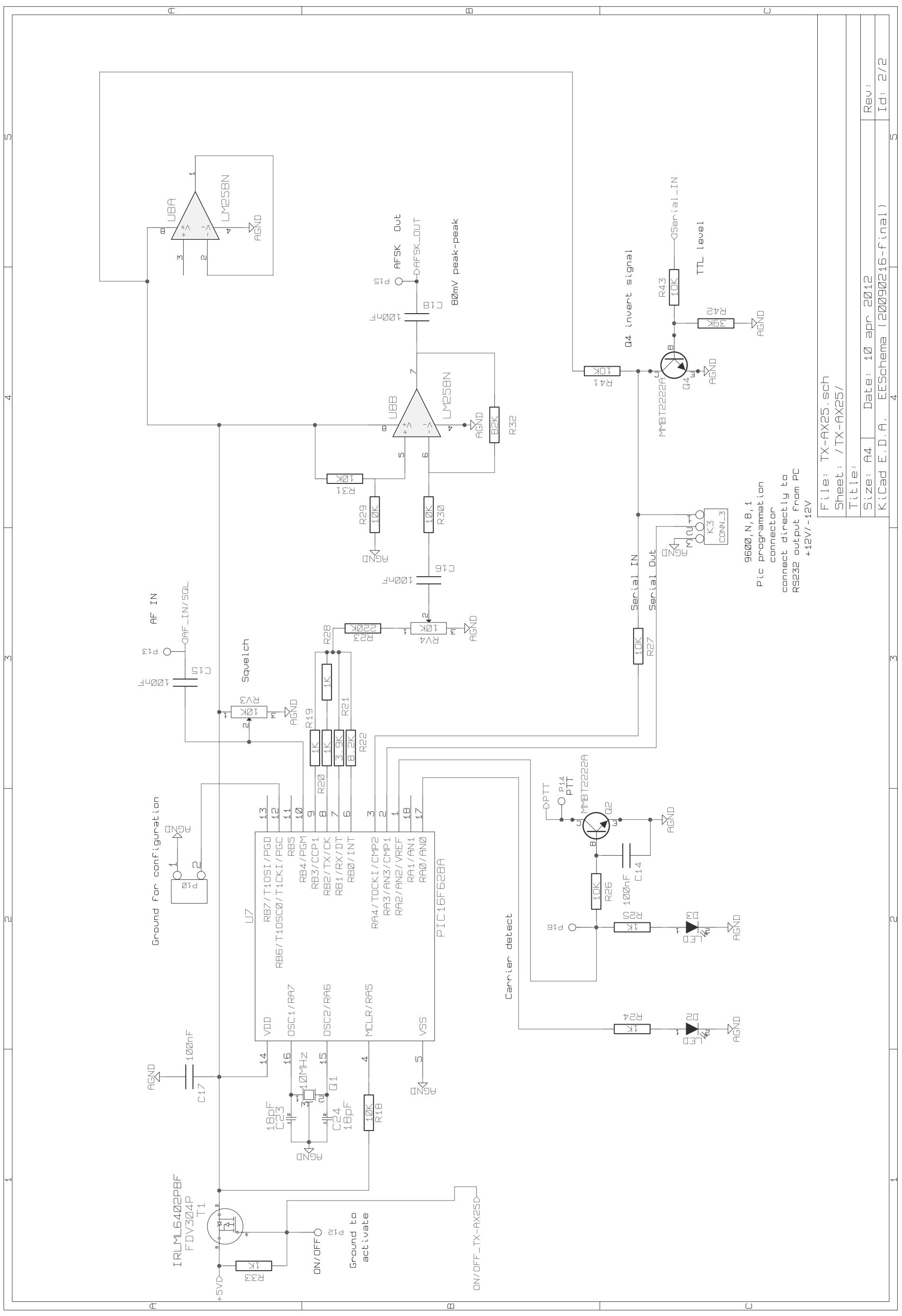


Figure II.11: Microcontroller schematic (continuation).

II.4.2 PCB Layout

II.4.2.1 Top Layer

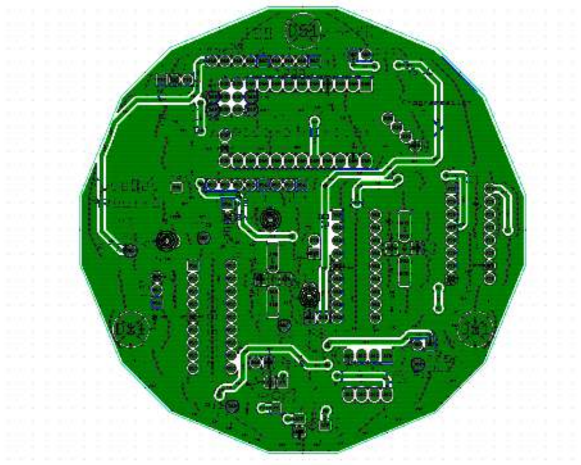


Figure II.12: Microcontroller PCB top layout.

II.4.2.2 Bottom Layer

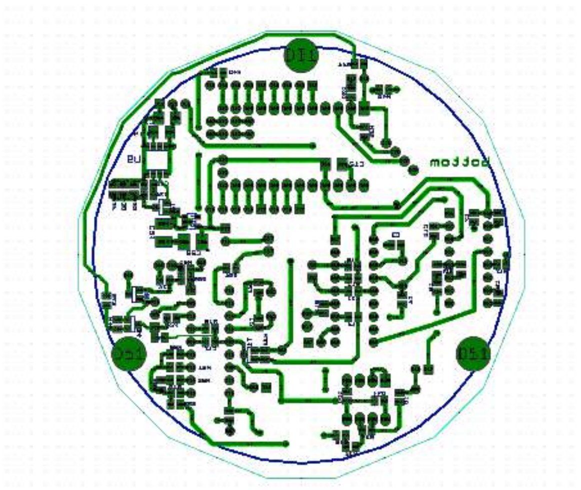


Figure II.13: Microcontroller PCB bottom layout.

II.4.3 List of Materials

PCB Ref	Value	Footprint	Observations	Part Number
C1	4.7uF	SM1206	16V	Farnell 922-7946
C3	18pF	SM0805	50V	Farnell 721-992
C4	18pF	SM0805	50V	Farnell 721-992
C6	100nF	SM0805	100V	FARNELL 1740681RL
C7	100nF	SM0805	100V	FARNELL 1740681RL
C9	100nF	SM0805	100V	FARNELL 1740681RL
C10	39pF	SM0805	50V	Farnell 722-030
C11	100nF	SM0805	100V	FARNELL 1740681RL
C12	4.7uF	SM1206	16V	Farnell 922-7946
C14	100nF	SM0805	100V	FARNELL 1740681RL
C15	100nF	SM0805	100V	FARNELL 1740681RL
C16	100nF	SM0805	100V	FARNELL 1740681RL
C17	100nF	SM0805	100V	FARNELL 1740681RL
C18	100nF	SM0805	100V	FARNELL 1740681RL
C19	4.7uF	SM1206	16V	Farnell 922-7946
C20	10nF	SM0805	100V	Farnell 1414661RL
C21	4.7uF	SM1210L	16V	Farnell 119-0116
C22	4.7uF	SM1210L	16V	Farnell 119-0116
C23	18pF	SM0805		
C24	18pF	SM0805		
D1	LED			Farnell 198-9923
D2	LED	LED-3MM		
D3	LED	LED-3MM		
K1	CONN_3			Farnell 973-1083
K2	CONN_3			Farnell 973-1083
K3	CONN_3	SIL-3		
P1	CONN_5			Farnell 973-1105
P2	CONN_4			Farnell 973-1091
P3	CONN_5			Farnell 973-1105
P5	CONN_2			Farnell 973-1601
P7	CONN_2			Farnell 973-1601
P8	CONN_2			Farnell 973-1601
P9	CONN_2			Farnell 973-1601
P10	CONN_2	SIL-2		
P17	CONN_2	SIL-2		Farnell 973-1601
P22	CONN_4	SIL-4		Farnell 973-1091
Q1	10MHz	QZ-cms		sparkfun COM-00541
Q2	MMBT2222A	SOT23EBC	Farnell 9846700	
Q3	3.579545MHz	QZ-cms	Farnell 153 8756	Farnell 164 0858
Q4	MMBT2222A	SOT23EBC	Farnell 9846700	
R5	10K	SM0805		Farnell 933-3720
R6	100K	SM0805		Farnell 933-3738
R7	10K	SM0805		Farnell 933-3738
R13	10K	SM0805		Farnell 933-3720
R14	10K	SM0805		Farnell 933-3720
R15	1K	SM0805		Farnell 933-3711
R16	10K	SM0805		Farnell 933-3720
R17	10K	SM0805		Farnell 933-3720
R18	10K	SM0805		Farnell 933-3720
R19	1K	SM0805		Farnell 933-3711
R20	1K	SM0805		Farnell 933-3711
R21	3.9K	SM0805		Farnell 933-4483
R22	8.2K	SM0805		Farnell 933-4904
R23	220K	SM0805		Farnell 933-4165
R24	1K	SM0805		Farnell 933-3711

R25	1K	SM0805		Farnell 933-3711
R26	1OK	SM0805		Farnell 933-3720
R27	1OK	SM0805		Farnell 933-3720
R28	1K	SM0805		Farnell 933-3711
R29	10K	SM0805		Farnell 933-3720
R30	10K	SM0805		Farnell 933-3720
R31	10K	SM0805		Farnell 933-3720
R32	82K	SM0805		
R33	1K	SM0805		Farnell 933-3711
R34	100K	SM0805		Farnell 933-3738
R35	R	SM0805		
R36	R	SM0805		
R37	R	SM0805		
R38	1K	SM0805		Farnell 933-3711
R39	1K	SM0805		Farnell 933-3711
R40	0 ohm	SM0805		
R41	1OK	SM0805		Farnell 933-3720
R42	39K	SM0805		
R43	1OK	SM0805		Farnell 933-3720
R44	0 ohm	SM0805		
R45	0 ohm	SM0805		
RV3	10K	POT_CMS		Farnell 1141362
RV4	10K	POT_CMS		Farnell 1141362
T1	FDV304P	SOT23GDS		Farnell 9846123
T2	FDV303N	SOT23GDS		Farnell 984-5020
U4	MX614			
U6	PIC16F627-04/P			Farnell 976-0288
U7	PIC16F628A			Farnell 9760423
U8	LM258N			Farnell 1750142
U9	LTC1153	SO8E		Farnell 1273503
U10	LP2981AIM5-5.0	SOT23-5		Farnell 9779957
U11	ARDUINO-MINI 04			

Table II.4: List of materials for the microcontroller PCB.

II.5 Payload

II.5.1 Schematic

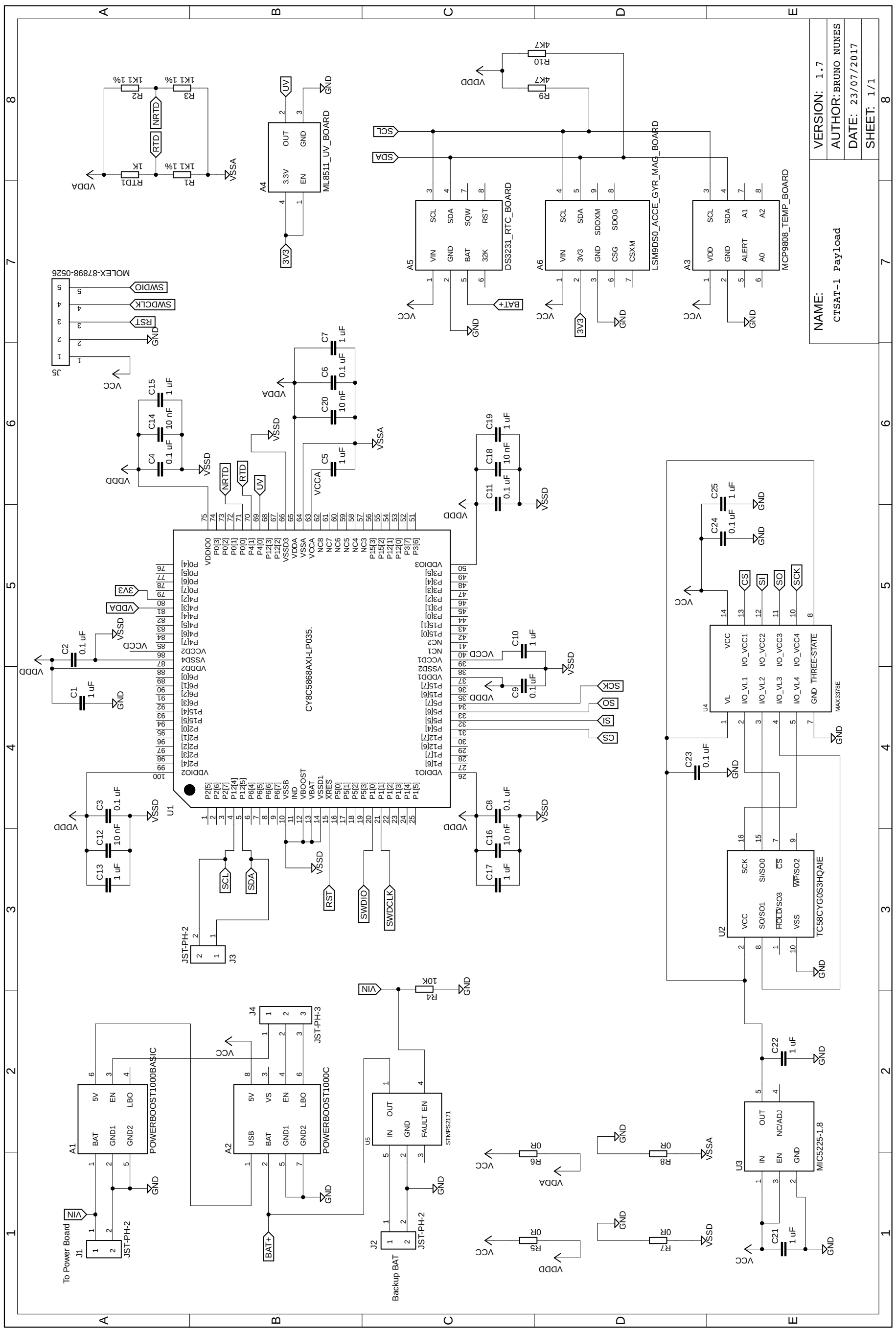


Figure II.14: CTSAT-1 payload schematic.

II.5.2 PCB Layout

II.5.2.1 Top Layer

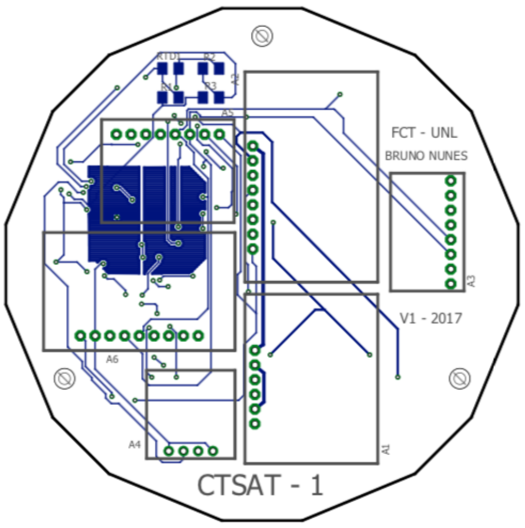


Figure II.15: CTSAT-1 payload PCB top layout.

II.5.2.2 Bottom Layer

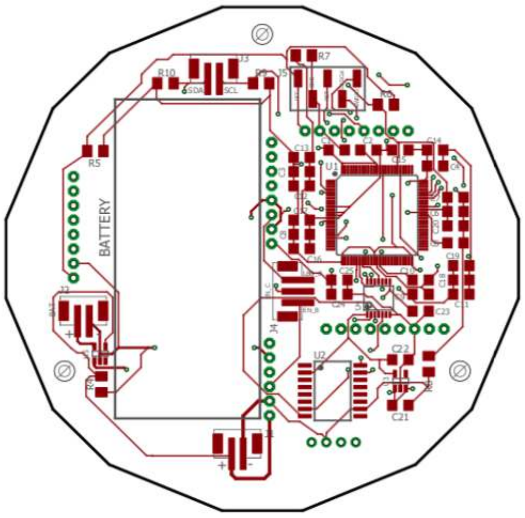


Figure II.16: CTSAT-1 payload PCB bottom layout.

II.5.3 List of Materials

PCB REF	VALUE	MANUFACTURER	PART NUMBER
U1	-	Cypress	CY8C5868AXI-LP035
U2	-	Toshiba	TC58CYG0S3HQAIE
U3	-	Microchip	MIC5225-1.8YM5-TR
U4	-	Maxim	MAX3378EEUD+
U5	-	STMicroelectronics	STMPS2171STR
RTD1	1K	Vishay	PTS1206M1B1K00P100
R1	1K1 1%	Vishay	CRCW12061K10FKEA
R2	1K1 1%	Vishay	CRCW12061K10FKEA
R3	1K1 1%	Vishay	CRCW12061K10FKEA
R4	10K 5%	Vishay	CRCW120610K0JNEA
R5	0R	Vishay	CRCW12060000Z0EAHP
R6	0R	Vishay	CRCW12060000Z0EAHP
R7	0R	Vishay	CRCW12060000Z0EAHP
R8	0R	Vishay	CRCW12060000Z0EAHP
R9	4K7 5%	Vishay	CRCW12064K70JNEA
R10	4K7 5%	Vishay	CRCW12064K70JNEA
C1	1 uF	Murata	GRM31CR71H105KA61L
C2	0.1 uF	Murata	GCM319R71H104KA37D
C3	0.1 uF	Murata	GCM319R71H104KA37D
C4	0.1 uF	Murata	GCM319R71H104KA37D
C5	1 uF	Murata	GRM31CR71H105KA61L
C6	0.1 uF	Murata	GCM319R71H104KA37D
C7	1 uF	Murata	GRM31CR71H105KA61L
C8	0.1 uF	Murata	GCM319R71H104KA37D
C9	0.1 uF	Murata	GCM319R71H104KA37D
C10	1 uF	Murata	GRM31CR71H105KA61L
C11	0.1 uF	Murata	GCM319R71H104KA37D
C12	0.01 uF	Murata	GRM319R71H102KA01J
C13	1 uF	Murata	GRM31CR71H105KA61L
C14	0.01 uF	Murata	GRM319R71H102KA01J
C15	1 uF	Murata	GRM31CR71H105KA61L
C16	0.01 uF	Murata	GRM319R71H102KA01J
C17	1 uF	Murata	GRM31CR71H105KA61L
C18	0.01 uF	Murata	GRM319R71H102KA01J
C19	1 uF	Murata	GRM31CR71H105KA61L
C20	0.01 uF	Murata	GRM319R71H102KA01J
C21	1 uF	Murata	GRM31CR71H105KA61L
C22	1 uF	Murata	GRM31CR71H105KA61L
C23	0.1 uF	Murata	GCM319R71H104KA37D
C24	0.1 uF	Murata	GCM319R71H104KA37D
C25	1 uF	Murata	GRM31CR71H105KA61L
J1	-	JST	B2B-PH-SM4-TB(LF)(SN)
J2	-	JST	B2B-PH-SM4-TB(LF)(SN)
J3	-	JST	B2B-PH-SM4-TB(LF)(SN)
J4	-	JST	B3B-PH-SM4-TB(LF)(SN)

J5	-	Molex	878980526
A1	-	Adafruit	2030
A2	-	Adafruit	2465
A3	-	Adafruit	1782
A4	-	Sparkfun	SEN-12705
A5	-	Adafruit	3013
A6	-	Adafruit	2021
BATTERY 3.7V 500mAh	-	Nimo	GSP 532248
Pin Headers for Modules	-	Samtec	TLW-140-06-T-S

Table II.5: List of materials for the CTSAT-1 payload PCB.

II.6 Solar Cells

II.6.1 Schematic

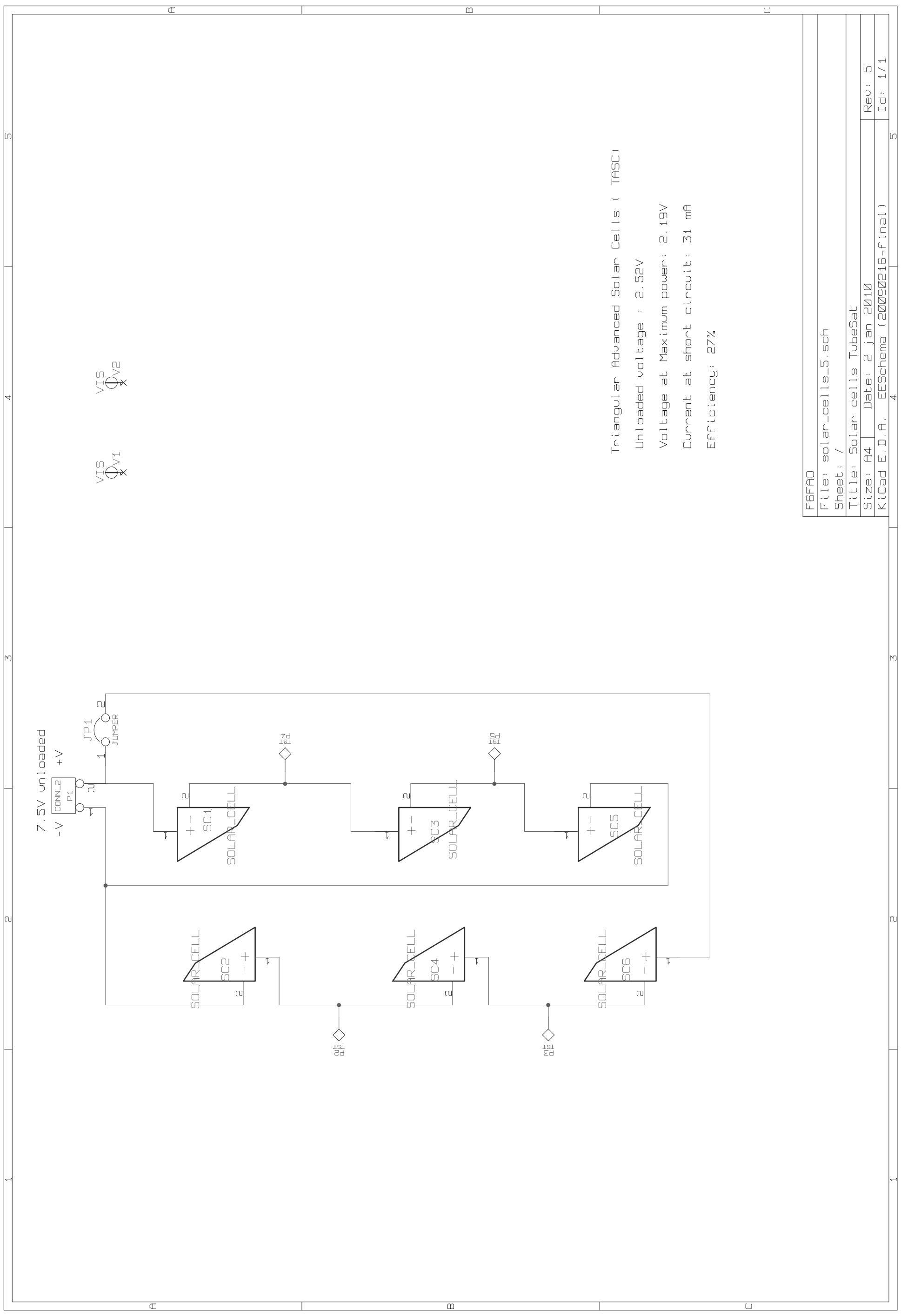


Figure II.17: Solar Cells schematic.

II.6.2 PCB Layout

II.6.2.1 Top Layer

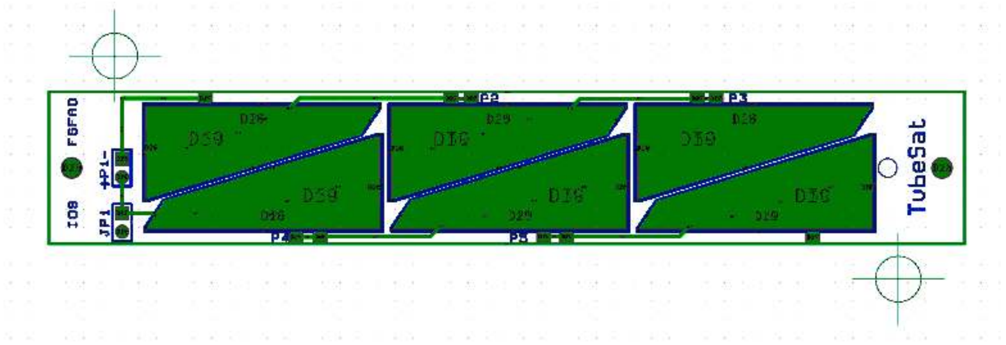


Figure II.18: Solar Cells PCB top layout.

II.6.2.2 Bottom Layer

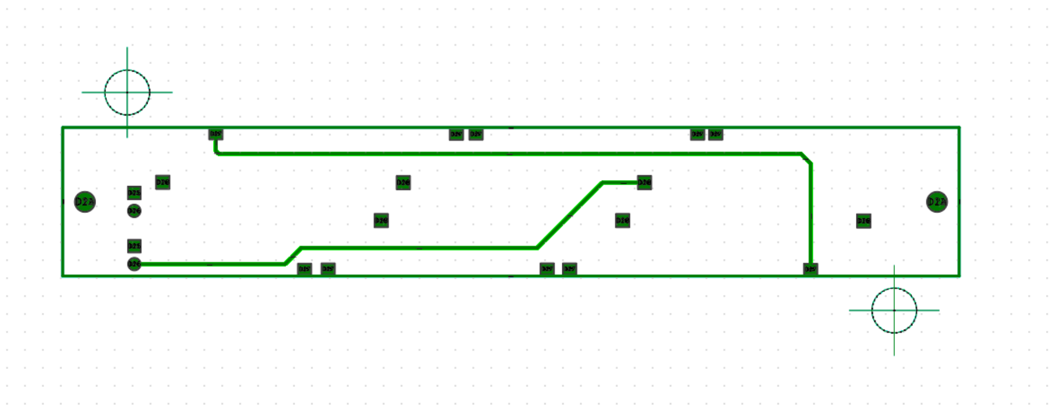


Figure II.19: Solar Cells PCB bottom layout.

Annex III

Testing Payload Components with Arduino - Technical Files

III.1 Arduino Testing Platform

III.1.1 Schematic

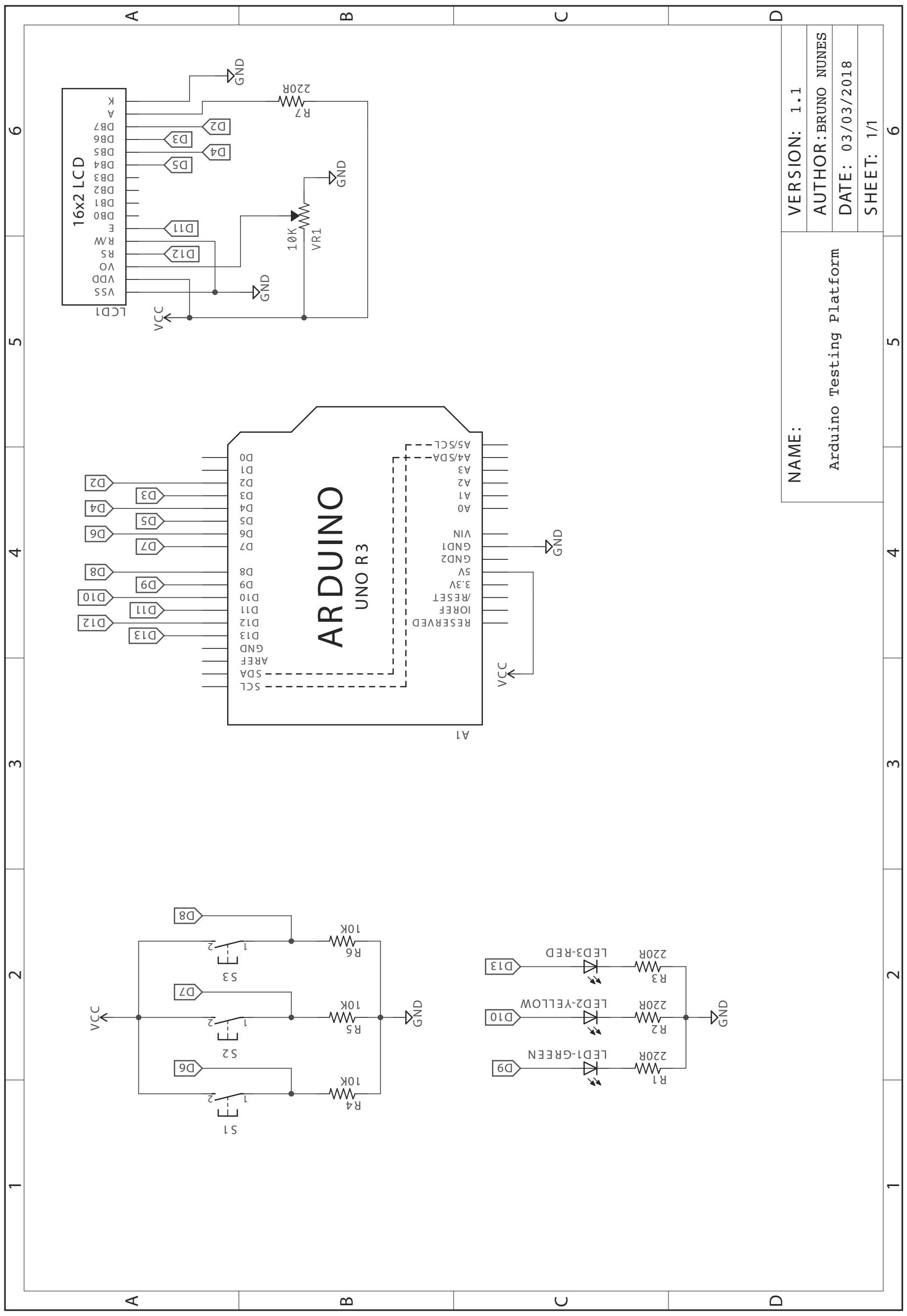


Figure III.1: Arduino testing platform schematic.

III.2 RTD Temperature Sensor

III.2.1 Schematic

III.2.1.1 Experiment 1 - USB

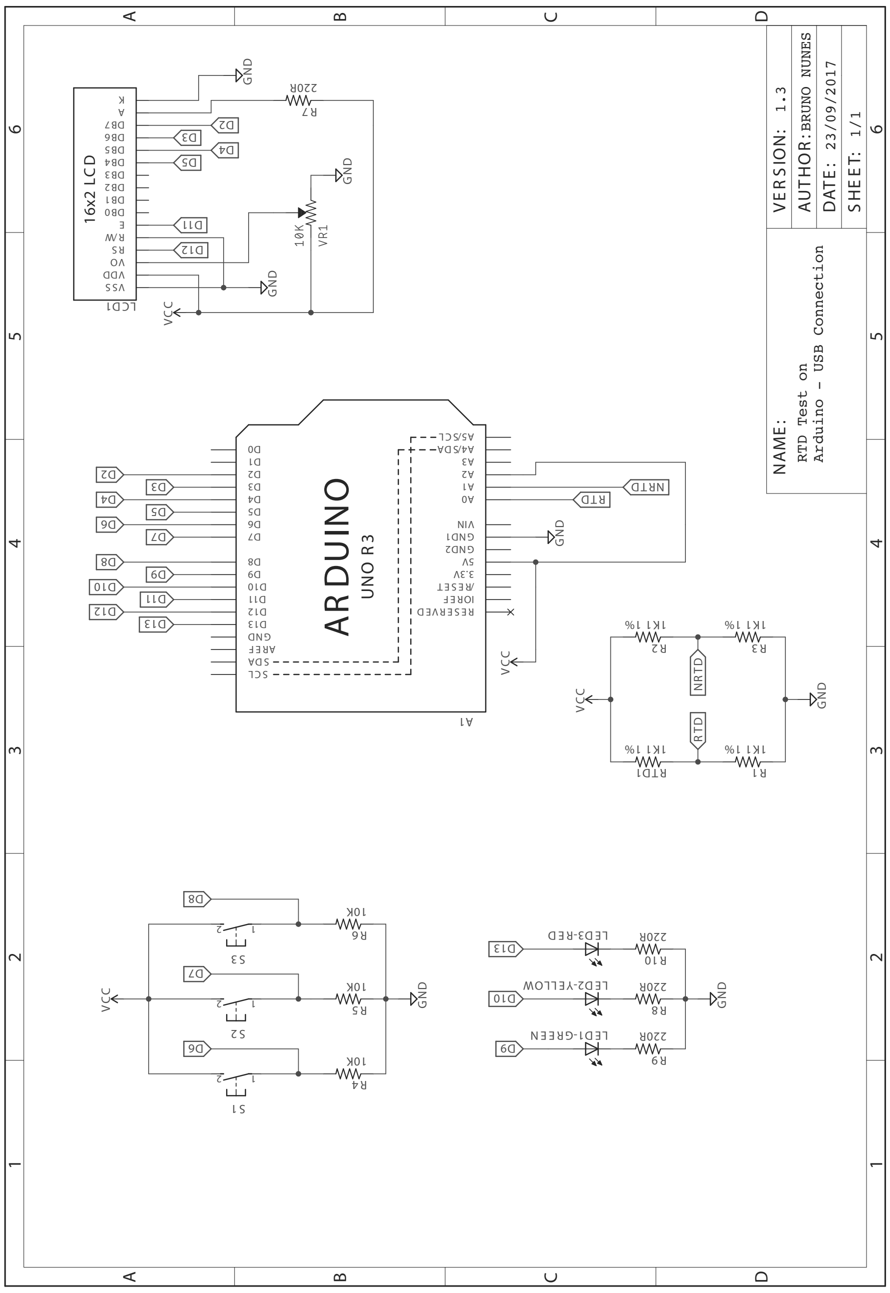


Figure III.2: Arduino RTD experiment 1 - USB schematic.

III.2.1.2 Experiment 2 - SD Card

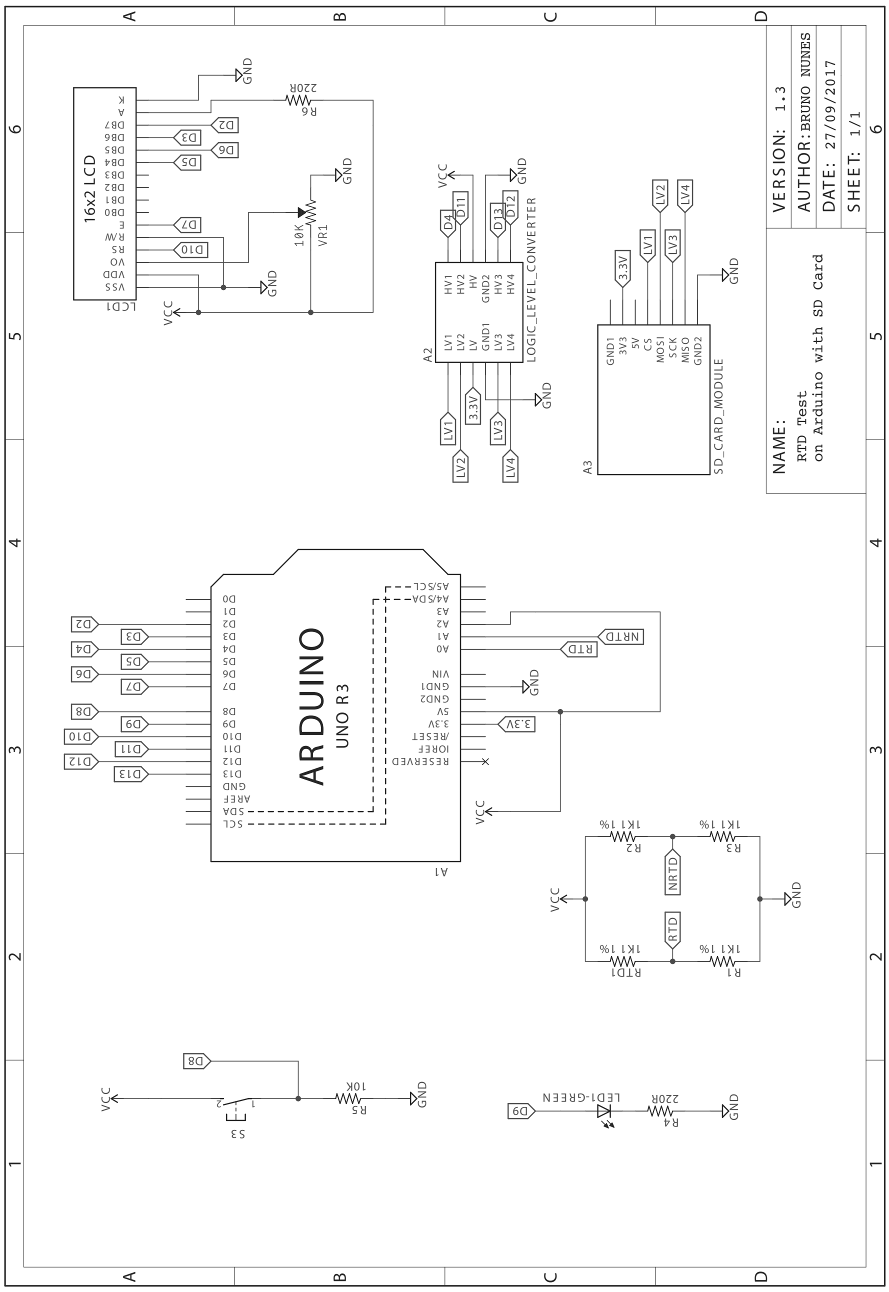


Figure III.3: Arduino RTD experiment 2 - SD Card schematic.

III.2.2 Source Code

III.2.2.1 Arduino

III.2.2.2 Experiment 1 - USB

```
/*
 *  RTD TEST - USB CONNECTION
 *
 *  Created : 27/06/2017
 *  Modified: 10/04/2018
 *
 *  Version: 1.2
 *
 */

#include <LiquidCrystal.h>

int RTD_pin = A0;
int nRTD_pin = A1;
int Vin_pin = A2;

float R0 = 1000.0;

//Exact values taken before soldering
float R1 = 1095.0;
float R2 = 1092.0;
float R3 = 1094.0;

int B1_pin = 6;
int B2_pin = 7;
int B3_pin = 8;

int LED_Green_pin = 9;
int LED_Yellow_pin = 10;
int LED_Red_pin = 13;

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);


void setup() {

  Serial.begin(9600); // Serial communication at 9600 bits per second

  lcd.begin(16, 2); //(columns, rows)

  pinMode (RTD_pin, INPUT);
  pinMode (nRTD_pin, INPUT);
  pinMode (Vin_pin, INPUT);

  pinMode (B1_pin, INPUT);
  pinMode (B2_pin, INPUT);
  pinMode (B3_pin, INPUT);

  pinMode (LED_Green_pin, OUTPUT);
  pinMode (LED_Yellow_pin, OUTPUT);
  pinMode (LED_Red_pin, OUTPUT);

}


void loop() {

  int RTD_level = average_analogRead(RTD_pin);
  int nRTD_level = average_analogRead(nRTD_pin);
  int Vin_level = average_analogRead(Vin_pin);

  //Convert vcc reading to voltage
  float vcc = Vin_level * 5.0;  // VCC +/- 5 Volts
  vcc = vcc / 1023.0;      // 2^10 (10 bits ADC)

  //5V power pin as a reference
  float output_voltageRTD = 5.0 / Vin_level * RTD_level;
  float output_voltagenRTD = 5.0 / Vin_level * nRTD_level;

  //From Wheatstone bridge circuit
  float RTD = ((R1*vcc)/(output_voltageRTD-output_voltagenRTD+((R3/(R2+R3))*
    vcc)))-R1;

  float RTD_method1 = RTD;

  float ratio_RT_R0 = RTD/R0;

  //Maps the value received by the sensor to the range described in datasheet
  float temperature_method2 = mapfloat(ratio_RT_R0, 0.78319, 1.66627, -55.0,
    175.0);

  Serial.print(RTD_method1);
  Serial.print(",");
  Serial.println(temperature_method2);

  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("RTD Temperature: ");
  lcd.setCursor(6, 1);
  lcd.print(temperature_method2);
  lcd.setCursor(12,1);
  lcd.print(char(223)); //print degree Symbol
  lcd.setCursor(13,1);
  lcd.print("C");

  delay(5000);

}


/*
 * Function to take the average of reading on a given pin
 */
int average_analogRead(int pin_ToRead)
{
  byte number_of_readings = 8;
  unsigned int current_value = 0;

  for(int x = 0 ; x < number_of_readings ; x++)
    current_value += analogRead(pin_ToRead);
  current_value /= number_of_readings;

  return(current_value);
}


/*
 * Function for mapping a value from one range to another range
 */
float mapfloat(float x, float in_min, float in_max, float out_min, float
    out_max)
{
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}
```

Listing III.1: Arduino RTD Experiment 1 Test Code

III.2.2.3 Experiment 2 - SD Card

```
/*
 *  RTD TEST - SD CARD
 *
 *  Created : 12/10/2017
 *  Modified: 14/04/2018
 *
 *  Version: 1.3
 *
 */

#include <SPI.h>
#include <SD.h>
#include <LiquidCrystal.h>

const int chipSelect = 4;
```

```

int RTD_pin = A0;
int nRTD_pin = A1;
int Vin_pin = A2;

float RTD = 0.0;

int R0 = 1000.0;

//Exact values taken before soldering
float R1 = 1095.0;
float R2 = 1092.0;
float R3 = 1094.0;

int B1_pin = 8; //button

int LED_Green_pin = 9;

float vcc = 0.0;
float voltage = 0.0;
float voltagebridge = 0.0;

LiquidCrystal lcd(10, 7, 5, 6, 3, 2);


void setup() {

  lcd.begin(16, 2); //(columns, rows)

  pinMode (RTD_pin, INPUT);
  pinMode (nRTD_pin, INPUT);
  pinMode (Vin_pin, INPUT);

  pinMode (B1_pin, INPUT);

  pinMode (LED_Green_pin, OUTPUT);

  //To make sure that the sd card is found
  if (!SD.begin(chipSelect)) {
    digitalWrite(LED_Green_pin, HIGH); // Lights up if the card failed or is
   not present
    return;
  }

}


void loop() {

  int RTD_level = average_analogRead(RTD_pin);
  int nRTD_level = average_analogRead(nRTD_pin);
  int Vin_level = average_analogRead(Vin_pin);

  //Convert vcc reading to voltage
  float vcc = Vin_level * 5.0;  // VCC +/- 5 Volts
  vcc = vcc / 1023.0;      // 2^10 (10 bits ADC)

  //5V power pin as a reference
  float output_voltageRTD = 5.0 / Vin_level * RTD_level;
  float output_voltagenRTD = 5.0 / Vin_level * nRTD_level;

  //From Wheatstone bridge circuit
  float RTD = ((R1*vcc)/(output_voltageRTD-output_voltagenRTD+((R3/(R2+R3))*
    vcc)))-R1;

  float RTD_method1 = RTD;

  float ratio_RT_R0 = RTD/R0;

  //Maps the value received by the sensor to the range described in datasheet
  float temperature_method2 = mapfloat(ratio_RT_R0, 0.78319, 1.66627, -55.0,
    175.0);

  File dataFile = SD.open("outputSD.txt", FILE_WRITE);

  //Check if the file is available
  if (dataFile) {
    dataFile.print(RTD_method1);
    dataFile.print(",");
    dataFile.println(temperature_method2);
    dataFile.close();
  }
  else {
    digitalWrite(LED_Green_pin, HIGH); // Lights up if there was an error
    opening the file
  }

  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("RTD Temperature: ");
  lcd.setCursor(6, 1);
  lcd.print(temperature_method2);
  lcd.setCursor(12,1);
  lcd.print(char(223)); //print degree Symbol
  lcd.setCursor(13,1);
  lcd.print("C");

  delay(5000);

}
```

```


/*
 * Function to take the average of reading on a given pin
 */
int average_analogRead(int pin_ToRead)
{
  byte number_of_readings = 8;
  unsigned int current_value = 0;

  for(int x = 0 ; x < number_of_readings ; x++)
    current_value += analogRead(pin_ToRead);
  current_value /= number_of_readings;

  return(current_value);
}


/*
 * Function for mapping a value from one range to another range
 */
float mapfloat(float x, float in_min, float in_max, float out_min, float
   out_max)
{
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}
```

Listing III.2: Arduino RTD Experiment 2 Test Code

III.2.2.4 Python

```
##############
## Script listens serial port and writes contents into a file
##############
##
##  Created : 09/10/2017
##  Modified: 16/05/2018
##
##  Version: 1.2
##

import serial;

serial_port = "/dev/cu.usbmodemFA131";
baud_rate = 9600; #In arduino, Serial.begin(baud_rate)
write_to_file_path = "output.txt";

output_file = open(write_to_file_path, "w+");
ser = serial.Serial(serial_port, baud_rate);
```

```

n = 1;
while n <= 360:
    line = ser.readline();
    line = line.decode("utf-8"); #ser.readline returns a binary, convert to
    string
    print(line);
    output_file.write(line);
    n = n + 1;

output_file.close();
```

Listing III.3: Script for Serial Communication

III.2.2.5 Matlab

```
%
% RTD TEST - Convert RTD resistance values into degrees Celsius
%
% Created : 19/09/2017
% Modified: 01/06/2018
%
% Version: 1.2
%

M = csvread('output.txt');

[rows, columns] = size(M);
n = 1;
j = rows;

R0 = 1000;

A = 3.9083*10^(-3);
B = -5.775*10^(-7);
C = -4.183*10^(-12);

S = zeros(rows,1);

while n <= j
    if M(n) < 1000 %Temperature from -55C to 0C
        RT = M(n);
        syms T_neg

        eq = R0 * (1 + A*T_neg + B*(T_neg^2) + C*(T_neg - 100) * T_neg^3) ==
    RT;

        S(n) = vpasolve(eq,T_neg, [-100 100]);
    elseif M(n) == 1000 %Temperature is 0C
```

```
        S(n) = 0;
    else %Temperature from 0C to +175C
        RT = M(n);
        S(n) = (- A + sqrt((A^2) - 4 * (B * ((1 - RT/R0))))) / (2*B);
    end
    n = n + 1;
end

fileID = fopen('results.txt','w');

fprintf(fileID,'%f\n',S);

fclose(fileID);
```

Listing III.4: Script for Convert RTD Values into Temperatures

III.3 Digital Temperature Sensor

III.3.1 Schematic

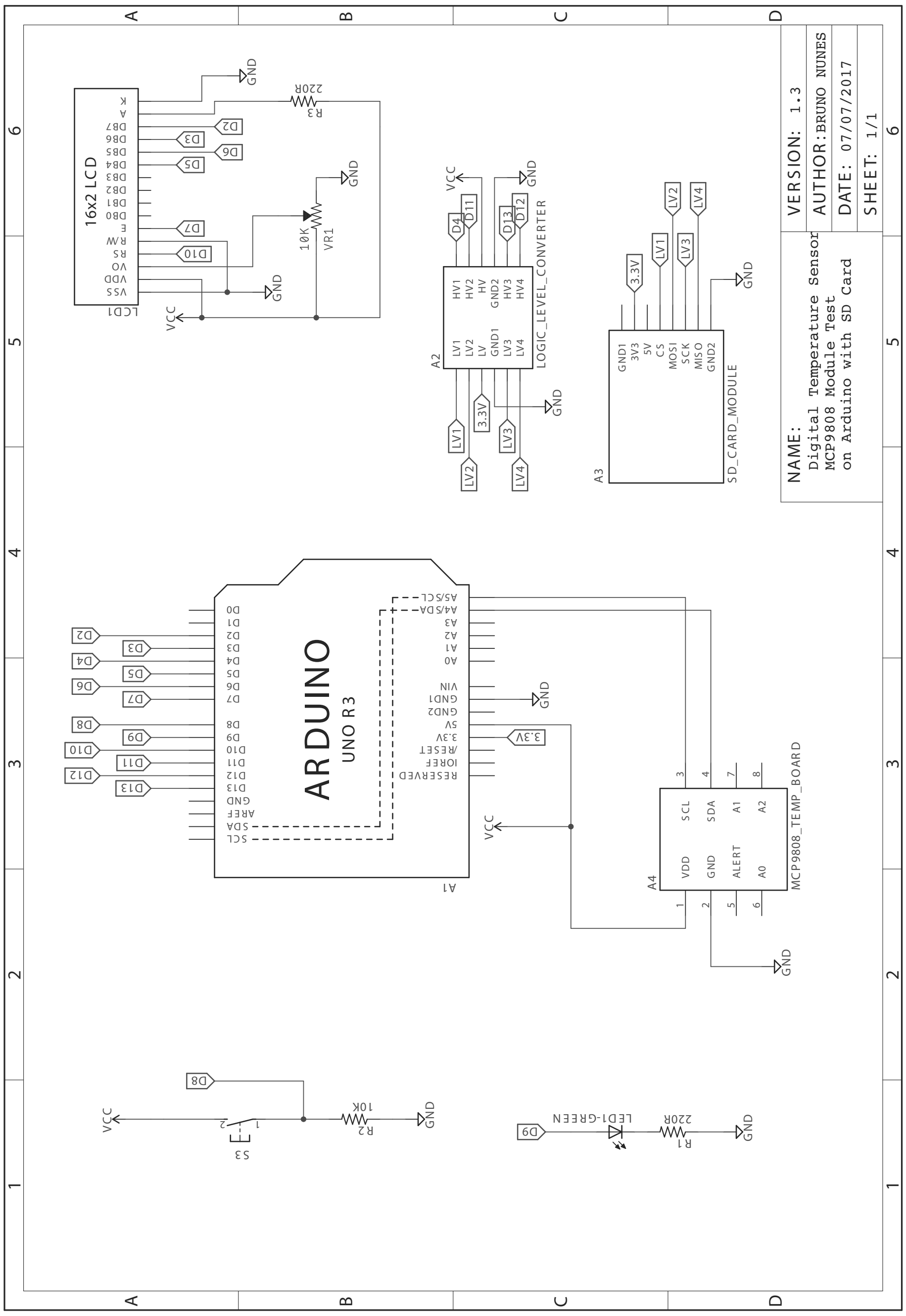


Figure III.4: Arduino digital temperature sensor test schematic.

III.3.2 Source Code

```
/*
 *  DIGITAL TEMPERATURE SENSOR TEST - Adafruit MCP9808 - SD CARD VERSION
 *
 *  Created : 27/06/2017
 *  Modified: 23/04/2018
 *
 *  Version: 1.6
 *
 *  based on https://learn.adafruit.com/adafruit-mcp9808-precision-i2c-temperature-sensor-guide/wiring
 *
 */

#include <SPI.h>
#include <SD.h>
#include <LiquidCrystal.h>
#include "Adafruit_MCP9808.h"

Adafruit_MCP9808 temp_sensor = Adafruit_MCP9808();

const int chipSelect = 4;

int B1_pin = 8; //button

int LED_Green_pin = 9;

LiquidCrystal lcd(10, 7, 5, 6, 3, 2);


void setup() {

  lcd.begin(16, 2);  //(columns, rows)

  pinMode (B1_pin, INPUT);

  pinMode (LED_Green_pin, OUTPUT);

  //To make sure that the sensor is found
  if (!temp_sensor.begin()) {
    digitalWrite(LED_Green_pin, HIGH); // Lights up if the sensor couldn't be found
    while (1);
  }

  //To make sure that the sd card is found
  if (!SD.begin(chipSelect)) {
    digitalWrite(LED_Green_pin, HIGH); // Lights up if the card failed or is not present
    return;
  }

}


void loop() {

  float temperature = temp_sensor.readTempC();

  File dataFile = SD.open("outputSD.txt", FILE_WRITE);

  //Check if the file is available
  if (dataFile) {
    dataFile.println(temperature);
    dataFile.close();
  }
  else {
    digitalWrite(LED_Green_pin, HIGH); // Lights up if there was an error
   opening the file
  }

  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Temperature: ");
  lcd.setCursor(6, 1);
  lcd.print(temperature);
  lcd.setCursor(12,1);
  lcd.print(char(223)); //print degree Symbol
  lcd.setCursor(13,1);
  lcd.print("C");

  delay(5000);

}
```

Listing III.5: Arduino Digital Temperature Sensor Test Code

III.4 UV Sensor

III.4.1 Schematic

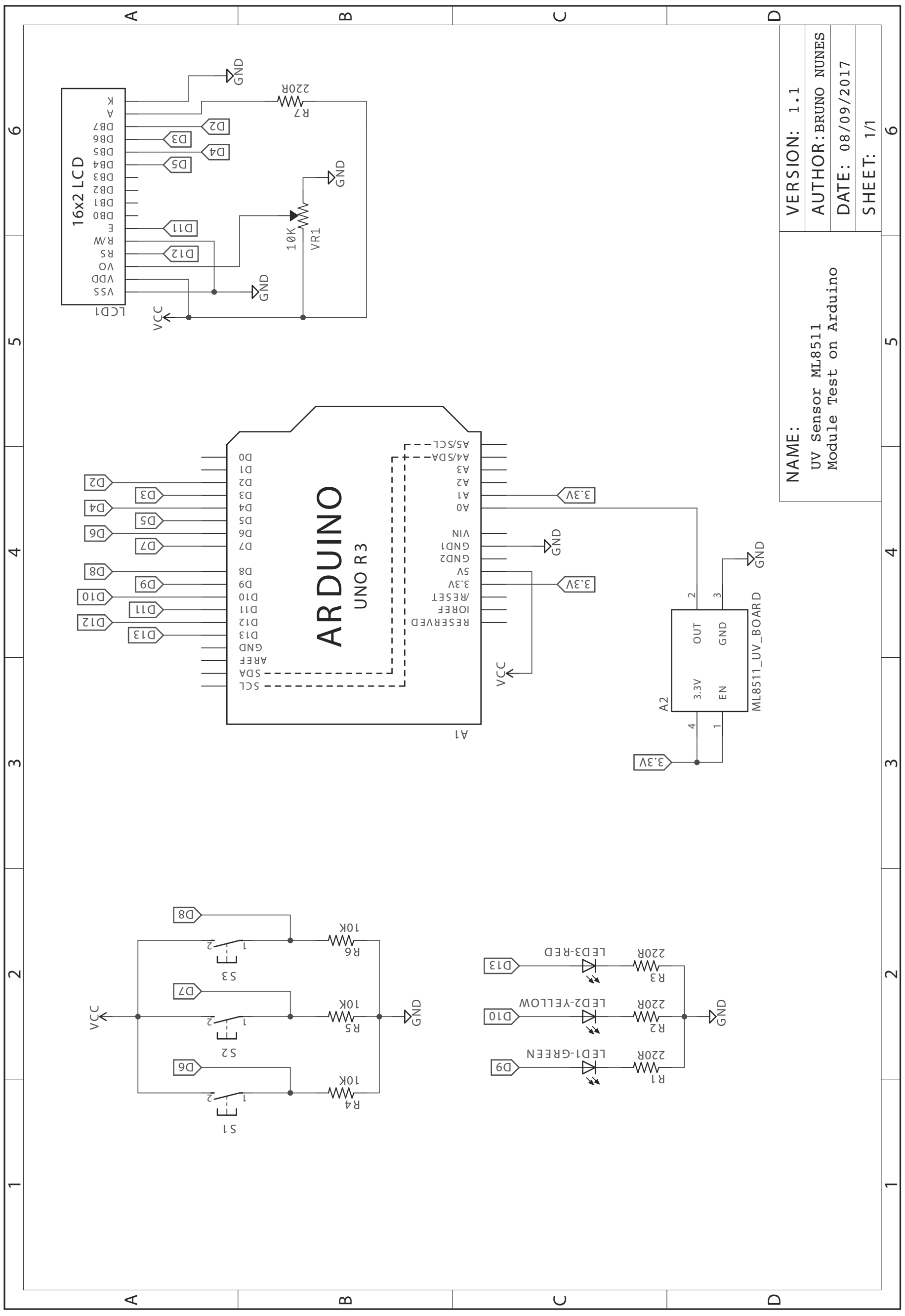


Figure III.5: Arduino UV sensor test schematic.

III.4.2 Source Code

```
/*
 * UV SENSOR TEST - Sparkfun ML8511
 *
 * Created : 26/06/2017
 * Modified: 19/04/2018
 *
 * Version: 1.4
 *
 * based on https://learn.sparkfun.com/tutorials/ml8511-uv-sensor-hookup-guide
 *
 */

#include <LiquidCrystal.h>

int UV_OUT_pin = A0; // Output
int REF_3V3_pin = A1; // Reference Pin 3.3V

int B1_pin = 6;
int B2_pin = 7;
int B3_pin = 8;

int LED_Green_pin = 9;
int LED_Yellow_pin = 10;
int LED_Red_pin = 13;

int state = 1;
int button_state = 0;

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);


void setup() {

  lcd.begin(16, 2); //(columns, rows)

  pinMode(UV_OUT_pin, INPUT);
  pinMode(REF_3V3_pin, INPUT);

  pinMode (B1_pin, INPUT);
  pinMode (B2_pin, INPUT);
  pinMode (B3_pin, INPUT);

  pinMode (LED_Green_pin, OUTPUT);
  pinMode (LED_Yellow_pin, OUTPUT);
  pinMode (LED_Red_pin, OUTPUT);

}
```

```


void loop() {

  button_state = digitalRead(B1_pin);

  if (button_state == HIGH){
    state++;
  }

  if (state == 3){
    state = 1;
  }

  int uv_level = average_analogRead(UV_OUT_pin);
  int ref_level = average_analogRead(REF_3V3_pin);

  //3.3V power pin as a reference to get a accurate output value from sensor
  float output_voltage = 3.3 / ref_level * uv_level;

  //Maps the value received by the sensor to the range described in datasheet
  float uv_intensity = mapfloat(output_voltage, 0.99, 2.9, 0.0, 15.0);

  lcd.clear();

  if (state == 1){
    lcd.setCursor(0,0);
    lcd.print("UV Intensity: ");
    lcd.setCursor(3, 1);
    lcd.print(uv_intensity);
    lcd.setCursor(9,1);
    lcd.print("mW/cm^2");
  }

   if (state == 2){
    lcd.setCursor(0,0);
    lcd.print("Out Voltage: ");
    lcd.setCursor(3, 1);
    lcd.print(output_voltage);
    lcd.setCursor(9,1);
    lcd.print("V");
  }

  delay(1000);

}

/*
 * Function to take the average of reading on a given pin
 */
```

```
int average_analogRead(int pin_ToRead)
{
  byte number_of_readings = 8;
  unsigned int current_value = 0;

  for(int x = 0 ; x < number_of_readings ; x++)
    current_value += analogRead(pin_ToRead);
  current_value /= number_of_readings;

  return(current_value);
}

/*
 * Function for mapping a value from one range to another range
 */
float mapfloat(float x, float in_min, float in_max, float out_min, float
    out_max)
{
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}
```

Listing III.6: Arduino UV Sensor Test Code

III.5 Magnetometer, Accelerometer and Gyroscope

III.5.1 Schematic

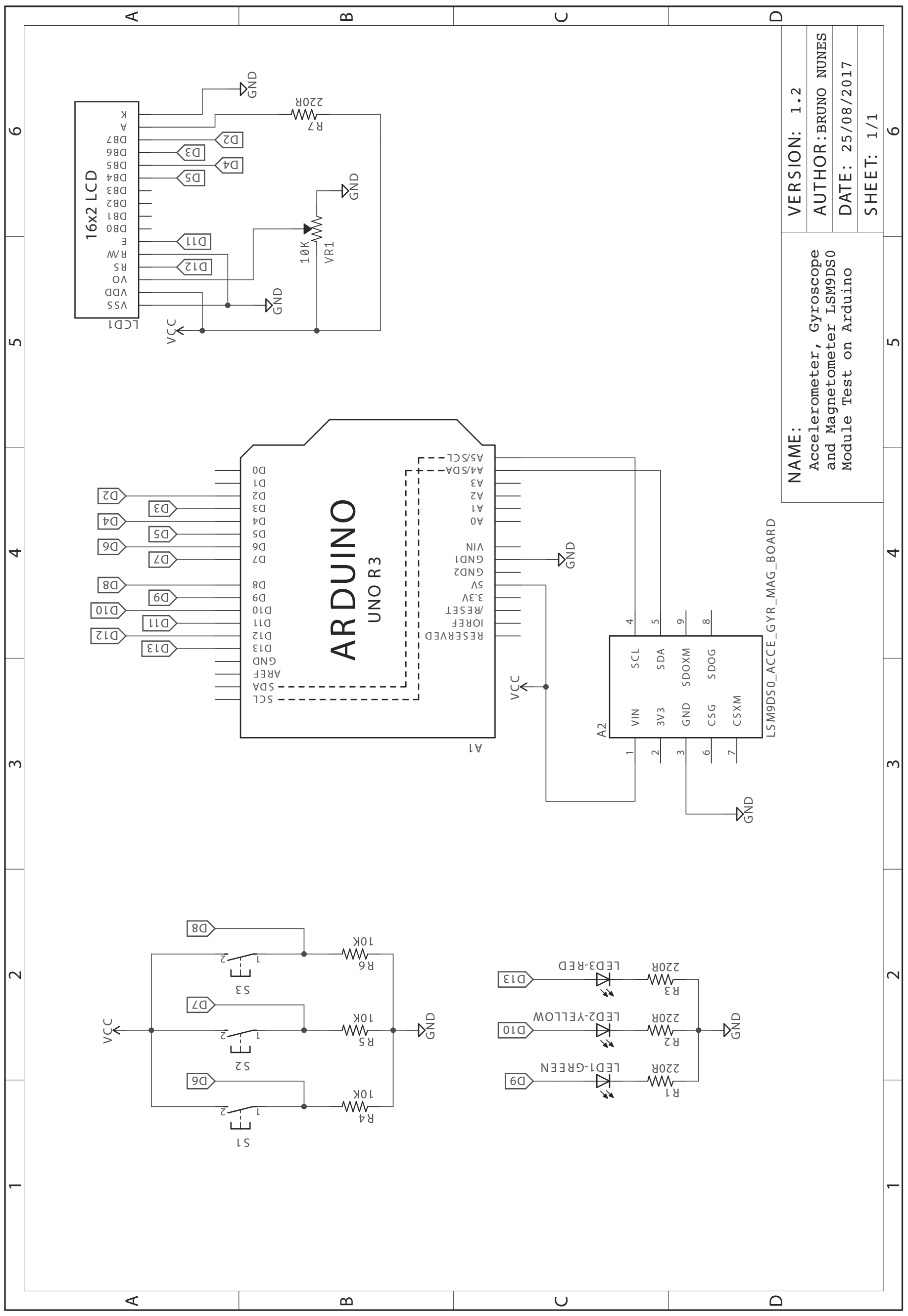


Figure III.6: Arduino Magnetometer, Accelerometer and Gyroscope test schematic.

III.5.2 Source Code

```
/*
 * MAGNETOMETER, ACCELEROMETER AND GYROSCOPE SENSOR TEST - Adafruit LSM9DS0
 *
 * Created : 26/06/2017
 * Modified: 07/05/2018
 *
 * Version 1.3
 *
 * based on https://learn.adafruit.com/adafruit-lsm9ds0-accelerometer-gyro-
    magnetometer-9-dof-breakouts/arduino-code
 *
 */

#include "Adafruit_LSM9DS0.h"
#include <LiquidCrystal.h>

Adafruit_LSM9DS0 sensor = Adafruit_LSM9DS0();

int B1_pin = 6;
int B2_pin = 7;
int B3_pin = 8;

int LED_Green_pin = 9;
int LED_Yellow_pin = 10;
int LED_Red_pin = 13;

int state = 1;
int button_state = 0;

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);


void configure_sensor()
{
  // 1.) Set the accelerometer range
  sensor.setupAccel(sensor.LSM9DS0_ACCELRANGE_2G);
  //sensor.setupAccel(sensor.LSM9DS0_ACCELRANGE_4G);
  //sensor.setupAccel(sensor.LSM9DS0_ACCELRANGE_6G);
  //sensor.setupAccel(sensor.LSM9DS0_ACCELRANGE_8G);
  //sensor.setupAccel(sensor.LSM9DS0_ACCELRANGE_16G);

  // 2.) Set the magnetometer sensitivity
  sensor.setupMag(sensor.LSM9DS0_MAGGAIN_2GAUSS);
  //sensor.setupMag(sensor.LSM9DS0_MAGGAIN_4GAUSS);
  //sensor.setupMag(sensor.LSM9DS0_MAGGAIN_8GAUSS);
  //sensor.setupMag(sensor.LSM9DS0_MAGGAIN_12GAUSS);

  // 3.) Setup the gyroscope
  sensor.setupGyro(sensor.LSM9DS0_GYROSCALE_245DPS);
  //sensor.setupGyro(sensor.LSM9DS0_GYROSCALE_500DPS);
  //sensor.setupGyro(sensor.LSM9DS0_GYROSCALE_2000DPS);
}


void setup() {

  lcd.begin(16, 2); //(columns, rows)

  pinMode (B1_pin, INPUT);
  pinMode (B2_pin, INPUT);
  pinMode (B3_pin, INPUT);

  pinMode (LED_Green_pin, OUTPUT);
  pinMode (LED_Yellow_pin, OUTPUT);
  pinMode (LED_Red_pin, OUTPUT);

  //To make sure that the sensor is found
  if (!sensor.begin())
  {
    digitalWrite(LED_Red_pin, HIGH); // Lights up if the sensor couldn't be
   found
    while (1);
  }

}


void loop() {

  sensor.read();

  button_state = digitalRead(B1_pin);

  if (button_state == HIGH){
    state++;
  }

  if (state == 4){
    state = 1;
  }

  lcd.clear();

  if (state == 1){
    lcd.setCursor(0,0);
    lcd.print("Accel X: ");
    lcd.setCursor(8,0);
    lcd.print(((sensor.accelData.x*0.061F)/1000)*9.80665F);
    lcd.setCursor(0, 1);
    lcd.print("Y: ");
    lcd.setCursor(2,1);
    lcd.print(((sensor.accelData.y*0.061F)/1000)*9.80665F);
    lcd.setCursor(9,1);
    lcd.print("Z: ");
    lcd.setCursor(11,1);
    lcd.print(((sensor.accelData.z*0.061F)/1000)*9.80665F);
  }

  if (state == 2){
    lcd.setCursor(0,0);
    lcd.print("Mag   X: ");
    lcd.setCursor(8,0);
    lcd.print((sensor.magData.x*0.08F)/1000);
    lcd.setCursor(0, 1);
    lcd.print("Y: ");
    lcd.setCursor(2,1);
    lcd.print((sensor.magData.y*0.08F)/1000);
    lcd.setCursor(9,1);
    lcd.print("Z: ");
    lcd.setCursor(11,1);
    lcd.print((sensor.magData.z*0.08F)/1000);
  }

  if (state == 3){
    lcd.setCursor(0,0);
    lcd.print("Gyro  X: ");
    lcd.setCursor(8,0);
    lcd.print(sensor.gyroData.x*0.00875F);
    lcd.setCursor(0, 1);
    lcd.print("Y: ");
    lcd.setCursor(2,1);
    lcd.print(sensor.gyroData.y*0.00875F);
    lcd.setCursor(9,1);
    lcd.print("Z: ");
    lcd.setCursor(11,1);
    lcd.print(sensor.gyroData.z*0.00875F);
  }

  delay(1000);

}
```

Listing III.7: Arduino Accelerometer, Magnetometer and Gyroscope Test Code

III.6 RTC

III.6.1 Schematic

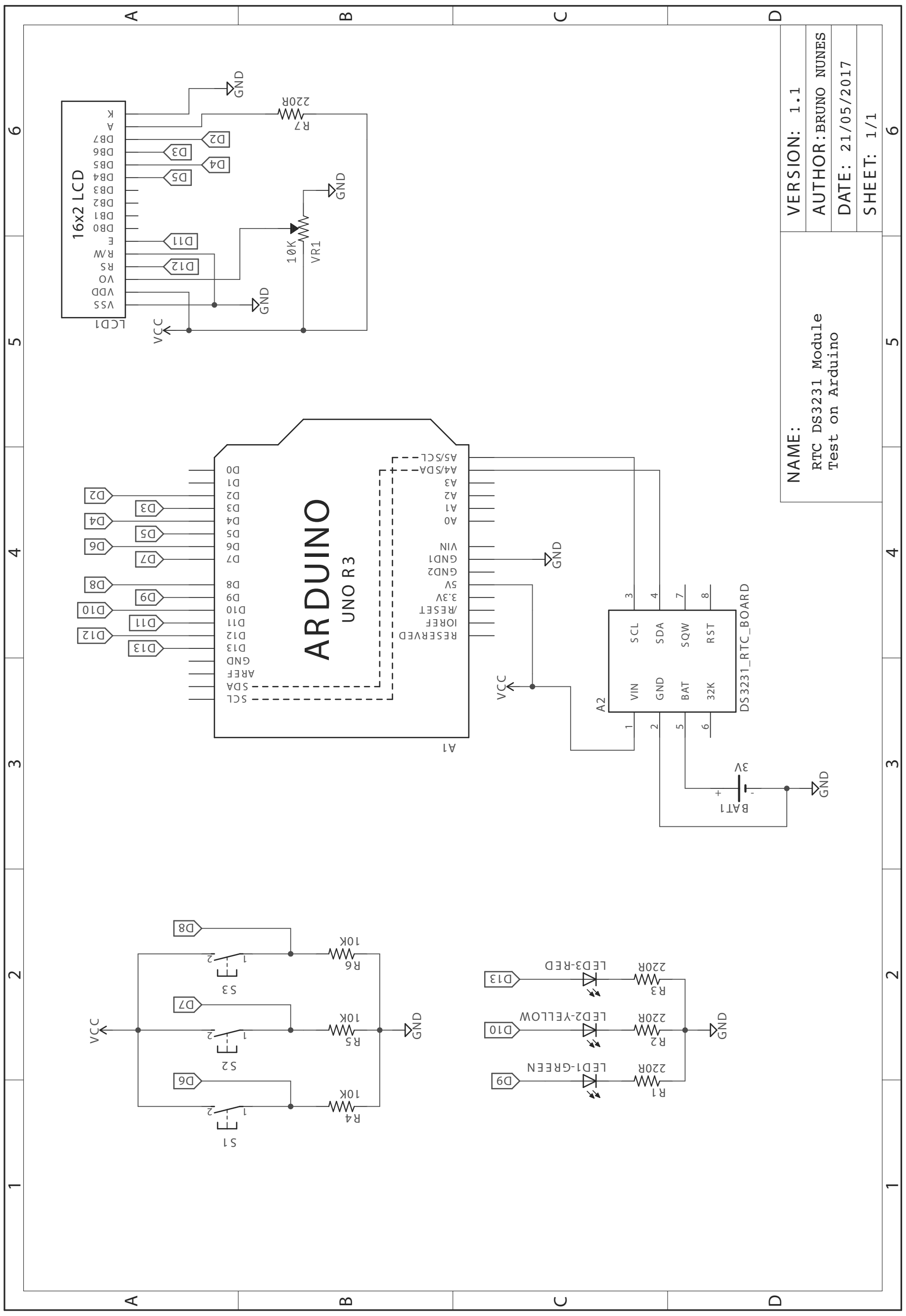


Figure III.7: Arduino RTC test schematic.

III.6.2 Source Code

```
/*
 * RTC TEST - Adafruit DS3231
 *
 * Created : 01/07/2017
 * Modified: 02/05/2018
 *
 * Version: 1.5
 *
 * based on https://learn.adafruit.com/adafruit-ds3231-precision-rtc-breakout
   /arduino-usage
 *
 */

#include "RTClib.h"
#include <LiquidCrystal.h>

RTC_DS3231 rtc;

int B1_pin = 6;
int B2_pin = 7;
int B3_pin = 8;

int LED_Green_pin = 9;
int LED_Yellow_pin = 10;
int LED_Red_pin = 13;

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);


void setup() {

  lcd.begin(16, 2); //(columns, rows)

  pinMode (B1_pin, INPUT);
  pinMode (B2_pin, INPUT);
  pinMode (B3_pin, INPUT);

  pinMode (LED_Green_pin, OUTPUT);
  pinMode (LED_Yellow_pin, OUTPUT);
  pinMode (LED_Red_pin, OUTPUT);

  //To make sure that the rtc is found
  if (! rtc.begin()) {
    digitalWrite(LED_Red_pin, HIGH); // Lights up if the rtc couldn't be found
    while (1);
  }

  if (rtc.lostPower()) {
    rtc.adjust(DateTime(F(__DATE__), F(__TIME__))); // sets the RTC to the date & time this sketch was compiled
    //rtc.adjust(DateTime(2014, 1, 21, 3, 0, 0)); // sets the RTC with an explicit date & time, for example January 21, 2014 at 3am
  }

}


void loop() {

  DateTime now = rtc.now();

  lcd.clear();

  lcd.setCursor(0,0);
  lcd.print("Date: ");
  lcd.setCursor(6,0);
  lcd.print(now.day());
  lcd.setCursor(8,0);
  lcd.print("/");
  lcd.setCursor(9,0);
  lcd.print(now.month());
  lcd.setCursor(11,0);
  lcd.print("/");
  lcd.setCursor(12,0);
  lcd.print(now.year());

  lcd.setCursor(0, 1);
  lcd.print("Time:");
  lcd.setCursor(6, 1);
  lcd.print(now.hour());
  lcd.setCursor(8,1);
  lcd.print(":");
  lcd.setCursor(9,1);
  lcd.print(now.minute());
  lcd.setCursor(11,1);
  lcd.print(":");
  lcd.setCursor(12,1);
  lcd.print(now.second());

  delay(1000);

}
```

Listing III.8: Arduino RTC Test Code

Annex IV

Testing Payload Components with PSoC - Technical Files

IV.1 PSoC Testing Platform

IV.1.1 Schematic

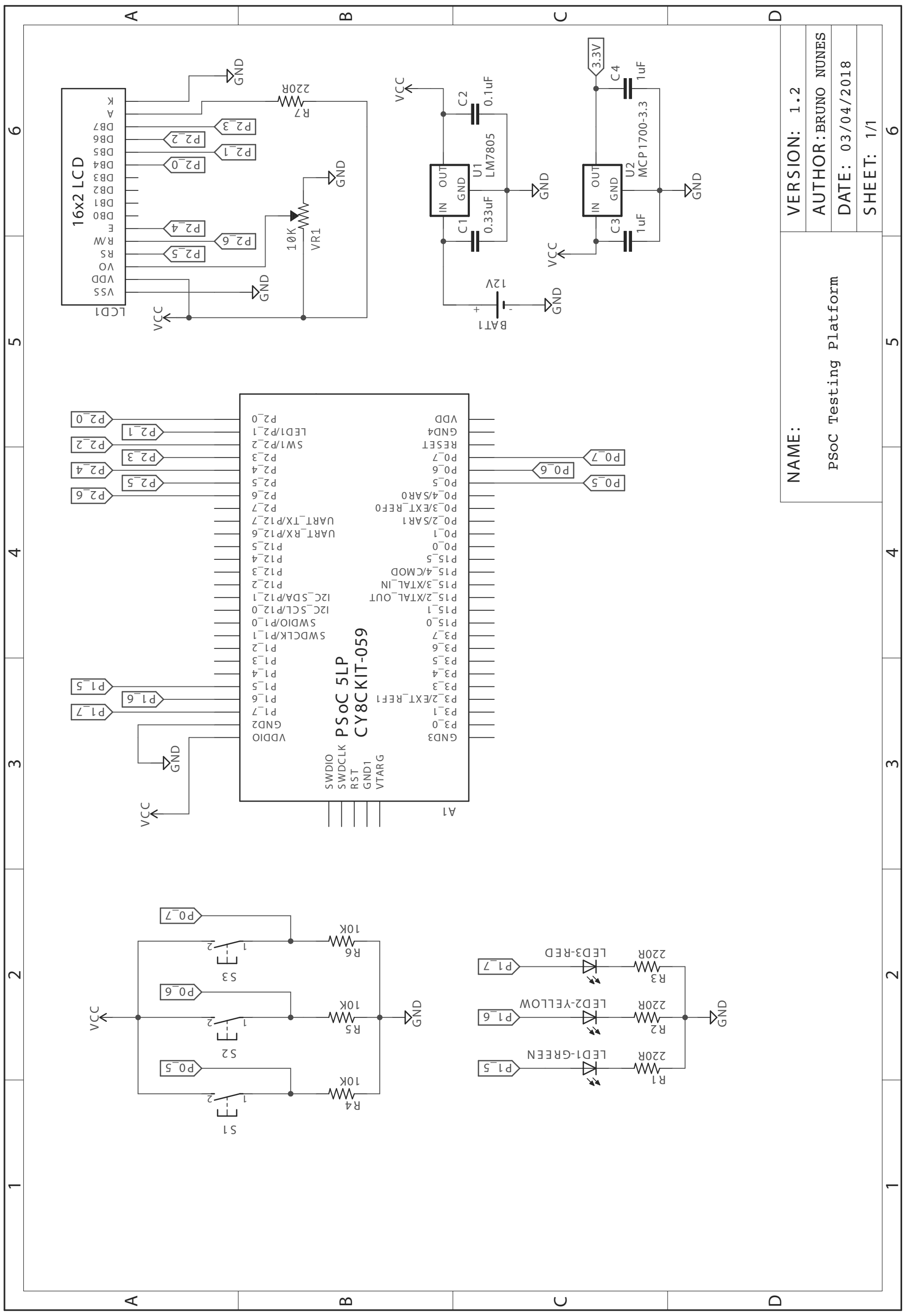


Figure IV.1: PSoC testing platform schematic.

IV.2 RTD Temperature Sensor

IV.2.1 Schematic

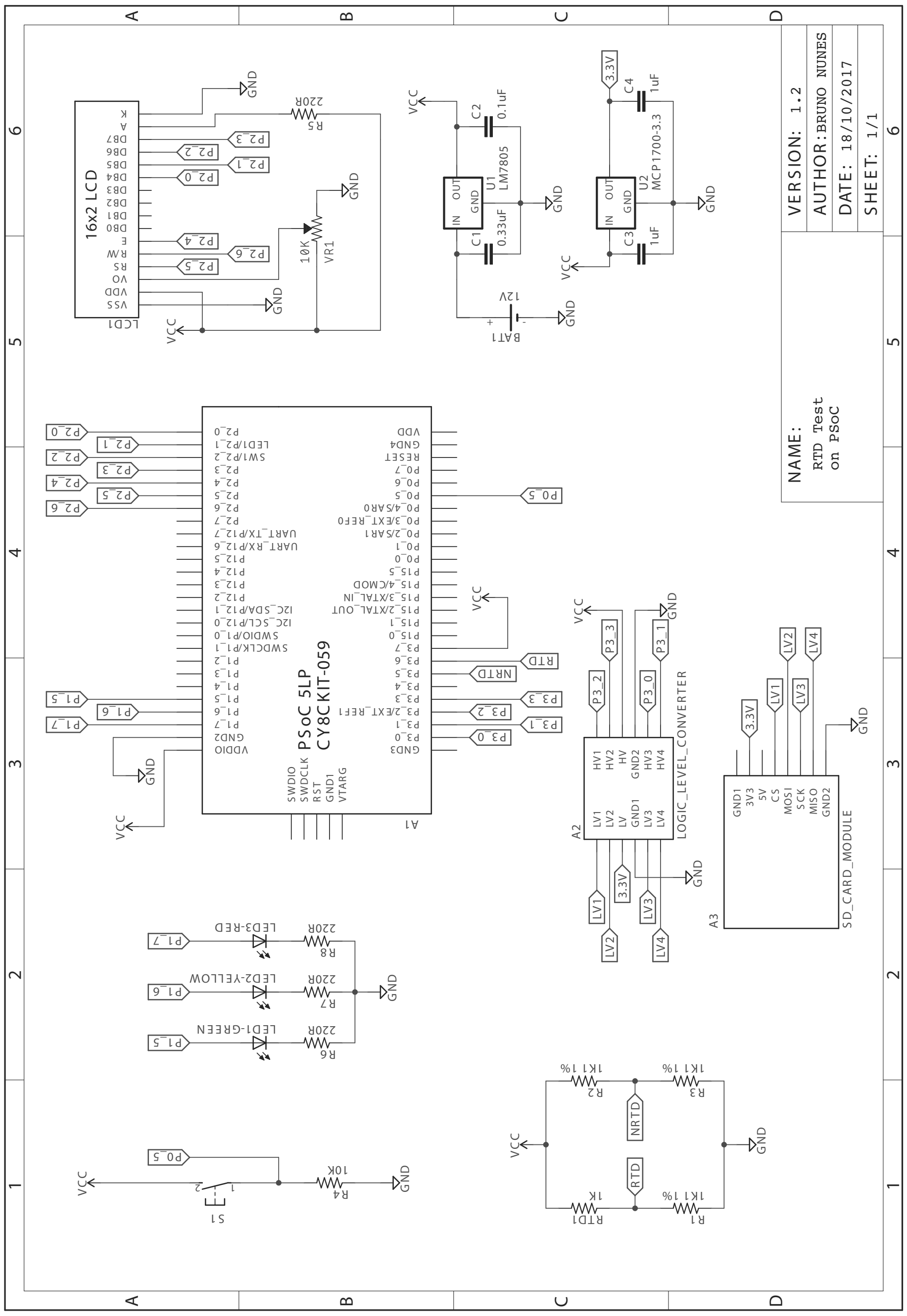


Figure IV.2: PSoC RTD test schematic.

IV.2.2 Source Code

```
/* ========================================
 *
 *  PSoC
 *  RTD TEST - SD CARD
 *
 *  Created : 12/01/2018
 *  Modified: 21/11/2018
 *
 *  Version: 1.6
 * ========================================
*/
#include "project.h"
#include "stdio.h"
#include <FS.h>

float mapfloat(float x, float in_min, float in_max, float out_min, float
    out_max);

int main(void)
{
    CyGlobalIntEnable; /* Enable global interrupts. */

    /* Start the components */
    LCD1_Start();
    ADC_DelSig1_Start();
    AMux1_Start();

    /* Initialize file system */
    FS_Init();

    int32 fromADC[3];

    int32 R0 = 1000;

    //Exact values taken before soldering
    float32 R1 = 1095.0;
    float32 R2 = 1092.0;
    float32 R3 = 1094.0;

    float32 vcc = 0.0;
    float32 RTD_voltage = 0.0;
    float32 nRTD_voltage = 0.0;

    char str_RTD_voltage[15];
    char str_nRTD_voltage[15];
    char str_vcc[15];

    char str_temperature_method1[15];
    char str_temperature_method2[15];

    char sdFile[11] = "Output.txt";

    FS_FILE * pFile;


    for(;;)
    {
        AMux1_FastSelect(0);
        ADC_DelSig1_StartConvert();
        ADC_DelSig1_IsEndConversion(ADC_DelSig1_WAIT_FOR_RESULT);
        fromADC[0] = ADC_DelSig1_GetResult32();
        ADC_DelSig1_StopConvert();

        RTD_voltage = ADC_DelSig1_CountsTo_Volts(fromADC[0]);
        sprintf(str_RTD_voltage,"%.3f",RTD_voltage);

        AMux1_FastSelect(1);
        ADC_DelSig1_StartConvert();
        ADC_DelSig1_IsEndConversion(ADC_DelSig1_WAIT_FOR_RESULT);
        fromADC[1] = ADC_DelSig1_GetResult32();
        ADC_DelSig1_StopConvert();

        nRTD_voltage = ADC_DelSig1_CountsTo_Volts(fromADC[1]);
        sprintf(str_nRTD_voltage,"%.3f",nRTD_voltage);

        AMux1_FastSelect(2);
        ADC_DelSig1_StartConvert();
        ADC_DelSig1_IsEndConversion(ADC_DelSig1_WAIT_FOR_RESULT);
        fromADC[2] = ADC_DelSig1_GetResult32();
        ADC_DelSig1_StopConvert();

        vcc = ADC_DelSig1_CountsTo_Volts(fromADC[2]);
        sprintf(str_vcc,"%.3f",vcc);

        float32 RTD = ((R1*vcc)/(RTD_voltage-nRTD_voltage+((R3/(R2+R3))*vcc)))
-R1;

        float32 RTD_method1 = RTD;
        sprintf(str_temperature_method1,"%.3f",RTD_method1);

        float32 ratio_RT_R0 = RTD/R0;

        //Maps the value received by the sensor to the range described in
datasheet
        float32 temperature_method2 = mapfloat(ratio_RT_R0, 0.78319, 1.66627,
-55.0, 175.0);
        sprintf(str_temperature_method2,"%.3f",temperature_method2);

```

```c
        pFile = FS_FOpen(sdFile, "a");

        if(pFile)
        {
            FS_Write(pFile, str_temperature_method1, 8u);
            FS_Write(pFile,",",1u);
            FS_Write(pFile,str_temperature_method2,7u);
            FS_Write(pFile,"\r\n",4u);

            FS_FClose(pFile);
        }
        else
        {
            LED_Red_Write(1u);// Lights up if there was an error opening the
    file
        }


        /* Clear LCD */
        LCD1_ClearDisplay();

        LCD1_Position(0,0);
        LCD1_PrintString("RTD Temperature: ");
        LCD1_Position(1,6);
        LCD1_PrintString(str_temperature_method2);
        LCD1_Position(1,12);
        LCD1_PutChar(223);
        LCD1_Position(1,13);
        LCD1_PrintString("C");

        CyDelay(5000u);
    }
}


/*
 * Function for mapping a value from one range to another range
 */
float mapfloat(float x, float in_min, float in_max, float out_min, float
    out_max)
{
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}

/* [] END OF FILE */
```

Listing IV.1: PSoC RTD Test Code

IV.3 Digital Temperature Sensor

IV.3.1 Schematic

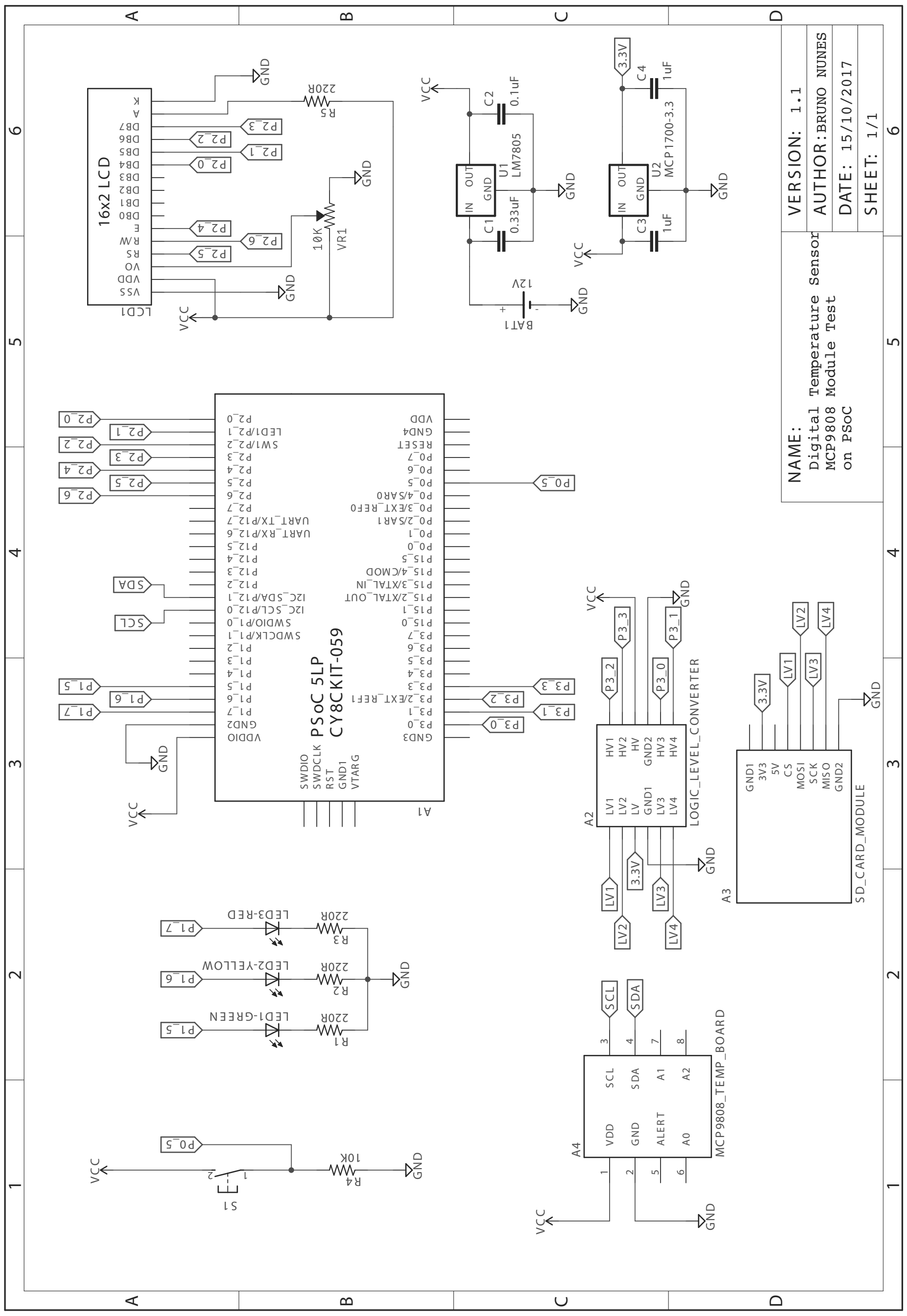


Figure IV.3: PSoC digital temperature sensor test schematic.

IV.3.2 Source Code

```
/* ========================================
 *
 *  PSoC
 *  DIGITAL TEMPERATURE SENSOR TEST - Adafruit MCP9808
 *
 *  Created : 05/02/2018
 *  Modified: 11/01/2019
 *
 *  Version: 1.8
 * ========================================
*/
#include "project.h"
#include "stdio.h"
#include <FS.h>


/* The I2C Slave address by default in a PSoC device is 8 */
#define MCP9808_I2C_ADDRESS     (0x18)
/* Set the write buffer length to be 16 bits or 2 bytes */
#define MCP9808_BUFFER_SIZE     (2u)

#define MCP9808_REG_AMBIENT_TEMP_ADRESS     (0x05)
#define MCP9808_REG_MANUF_ID_ADRESS         (0x06)
#define MCP9808_REG_DEVICE_ID_ADRESS        (0x07)

#define MCP9808_MANUFACTURER_ID     (0x0054)
#define MCP9808_DEVICE_ID           (0x0400)


int mcp9808_begin();
float32 get_temp(void);


int main(void)
{
    CyGlobalIntEnable; /* Enable global interrupts. */

    /* Start the components */
  LCD1_Start();
    I2C1_Start();

    /* Initialize file system */
    FS_Init();

    char str_temperature[15];

    char sdFile[11] = "Output.txt";

    FS_FILE * pFile;

    float32 temperature = 0.0;

    //make sure the sensor is found
    if (!mcp9808_begin())
    {
        LED_Yellow_Write(1u); // Lights up if the sensor couldn't be found
        while(1);
    }

    for(;;)
    {


        temperature = get_temp();
        sprintf(str_temperature,"%.3f",temperature);


        pFile = FS_FOpen(sdFile, "a");

        if(pFile)
        {
            FS_Write(pFile, str_temperature, 7u);
            FS_Write(pFile,"\r\n",4u);

            FS_FClose(pFile);
        }
        else
        {
            LED_Red_Write(1u);// Lights up if there was an error opening the
    file
        }

        /* Clear LCD */
        LCD1_ClearDisplay();

        LCD1_Position(0,0);
        LCD1_PrintString("Temperature: ");
        LCD1_Position(1,6);
        LCD1_PrintString(str_temperature);
        LCD1_Position(1,12);
        LCD1_PutChar(223);
        LCD1_Position(1,13);
        LCD1_PrintString("C");

        CyDelay(5000u);

    }

```

```
}

/*
Function to make sure that the mcp9808 is found
*/
int mcp9808_begin(void)
{
    int8_t buffer[MCP9808_BUFFER_SIZE];
    uint8_t upperByte;
    uint8_t lowerByte;
    int16_t manufacturerID;
    int16_t deviceID;

    //Initialize Transaction for Writing
    I2C1_MasterSendStart(MCP9808_I2C_ADDRESS,I2C1_WRITE_XFER_MODE);

    //Indicate Register to Write to
    I2C1_MasterWriteByte(MCP9808_REG_MANUF_ID_ADRESS);
    I2C1_MasterSendRestart(MCP9808_I2C_ADDRESS,I2C1_READ_XFER_MODE);

    //Read from Register (2 Bytes, last byte NAKed)
    buffer[0] = I2C1_MasterReadByte(I2C1_ACK_DATA);
    buffer[1] = I2C1_MasterReadByte(I2C1_NAK_DATA);

    //End Transaction
    I2C1_MasterSendStop();


    upperByte = buffer[0];
    lowerByte = buffer[1];

    manufacturerID = upperByte;
    manufacturerID <<= 8;
    manufacturerID |= lowerByte;

    //Initialize Transaction for Writing
    I2C1_MasterSendStart(MCP9808_I2C_ADDRESS,I2C1_WRITE_XFER_MODE);

    //Indicate Register to Write to
    I2C1_MasterWriteByte(MCP9808_REG_DEVICE_ID_ADRESS);
    I2C1_MasterSendRestart(MCP9808_I2C_ADDRESS,I2C1_READ_XFER_MODE);

    //Read from Register (2 Bytes, last byte NAKed)
    buffer[0] = I2C1_MasterReadByte(I2C1_ACK_DATA);
    buffer[1] = I2C1_MasterReadByte(I2C1_NAK_DATA);

    //End Transaction
    I2C1_MasterSendStop();


    upperByte = buffer[0];
    lowerByte = buffer[1];

    deviceID = upperByte;
    deviceID <<= 8;
    deviceID |= lowerByte;

    if(manufacturerID != MCP9808_MANUFACTURER_ID)
        return 0;

    if(deviceID != MCP9808_DEVICE_ID)
        return 0;

    return 1;
}
/*

Function to return the temperature
*/

float32 get_temp(void)
{
    float32 temperature;
    int8_t buffer[MCP9808_BUFFER_SIZE];
    uint8_t upperByte;
    uint8_t lowerByte;

    //Initialize Transaction for Writing
    I2C1_MasterSendStart(MCP9808_I2C_ADDRESS,I2C1_WRITE_XFER_MODE);

    //Indicate Register to Write to
    I2C1_MasterWriteByte(MCP9808_REG_AMBIENT_TEMP_ADRESS);
    I2C1_MasterSendRestart(MCP9808_I2C_ADDRESS,I2C1_READ_XFER_MODE);

    //Read from Register (2 Bytes, last byte NAKed)
    buffer[0] = I2C1_MasterReadByte(I2C1_ACK_DATA);
    buffer[1] = I2C1_MasterReadByte(I2C1_NAK_DATA);

    //End Transaction
    I2C1_MasterSendStop();


    upperByte = buffer[0];
    lowerByte = buffer[1];

    upperByte &= 0x01F; //Clear flag bits

    if((upperByte & 0x10) == 0x10) //Temperature < 0 C
    {
        upperByte &= 0x0F; //Clear SIGN bit
```

```
            float32 upperByte2 = upperByte;
            float32 lowerByte2 = lowerByte;
            temperature = -(256 -((upperByte2 * 16) + (lowerByte2 / 16)));
        }
        else    //Termperature >= 0 C
        {
            float32 upperByte2 = upperByte;
            float32 lowerByte2 = lowerByte;
            temperature = (upperByte2 * 16) + (lowerByte2 / 16);
        }

        return temperature;

}
/* [] END OF FILE */
```

Listing IV.2: PSoC Digital Temperature Sensor Test Code

IV.4 UV Sensor

IV.4.1 Schematic

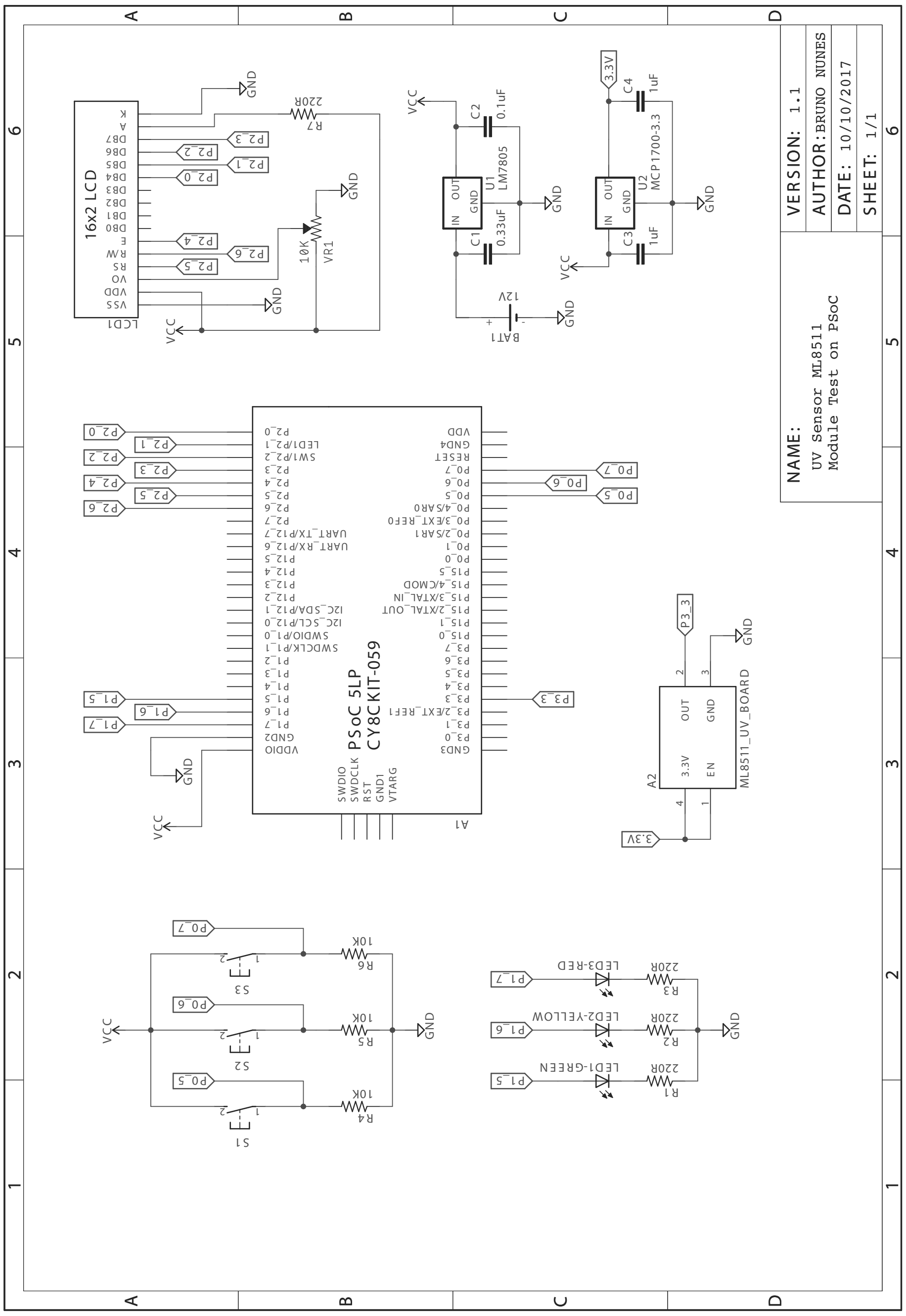


Figure IV.4: PSoC UV sensor test schematic.

IV.4.2 Source Code

```
/* ========================================
 *
 *  PSoC
 *  UV SENSOR TEST - Sparkfun ML8511
 *
 *  Created : 19/01/2018
 *  Modified: 03/12/2018
 *
 *  Version: 1.3
 * ========================================
*/
#include "project.h"
#include "stdio.h"

float mapfloat(float x, float in_min, float in_max, float out_min, float
    out_max);

int main(void)
{
    CyGlobalIntEnable; /* Enable global interrupts. */

    /* Start the components */
  LCD1_Start();
    ADC_DelSig1_Start();

    uint8_t state = 1;

    int32 fromADC;
    float32 uv_intensity = 0.0;
    float32 uv_voltage = 0.0;

    char str_uv_voltage[15];
    char str_uv_intensity[15];


    for(;;)
    {
        if(Button1_Read())
            state++;

        if (state == 3){
            state = 1;
        }

        ADC_DelSig1_StartConvert();
        ADC_DelSig1_IsEndConversion(ADC_DelSig1_WAIT_FOR_RESULT);
        fromADC = ADC_DelSig1_GetResult32();
        ADC_DelSig1_StopConvert();
```

```c

        uv_voltage = ADC_DelSig1_CountsTo_Volts(fromADC);
        sprintf(str_uv_voltage,"%.3f",uv_voltage);


        //Maps the value received by the sensor to the range described in
    datasheet
        uv_intensity = mapfloat(uv_voltage, 0.99, 2.9, 0.0, 15.0);
        sprintf(str_uv_intensity,"%.3f",uv_intensity);

        /* Clear LCD */
        LCD1_ClearDisplay();

        if (state == 1)
        {
            LCD1_Position(0,0);
            LCD1_PrintString("UV Intensity");
            LCD1_Position(1,3);
            LCD1_PrintString(str_uv_intensity);
            LCD1_Position(1,9);
            LCD1_PrintString("mw/cm^2");

        }

         if (state == 2)
        {
            LCD1_Position(0,0);
            LCD1_PrintString("Out Voltage: ");
            LCD1_Position(1,3);
            LCD1_PrintString(str_uv_voltage);
            LCD1_Position(1,9);
            LCD1_PrintString("V");

        }

        CyDelay(3000u);

    }
}

/*
 * Function for mapping a value from one range to another range
 */
float mapfloat(float x, float in_min, float in_max, float out_min, float
    out_max)
{
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}

/* [] END OF FILE */
```

Listing IV.3: PSoC UV Sensor Test Code

IV.5 Magnetometer, Accelerometer and Gyroscope

IV.5.1 Schematic

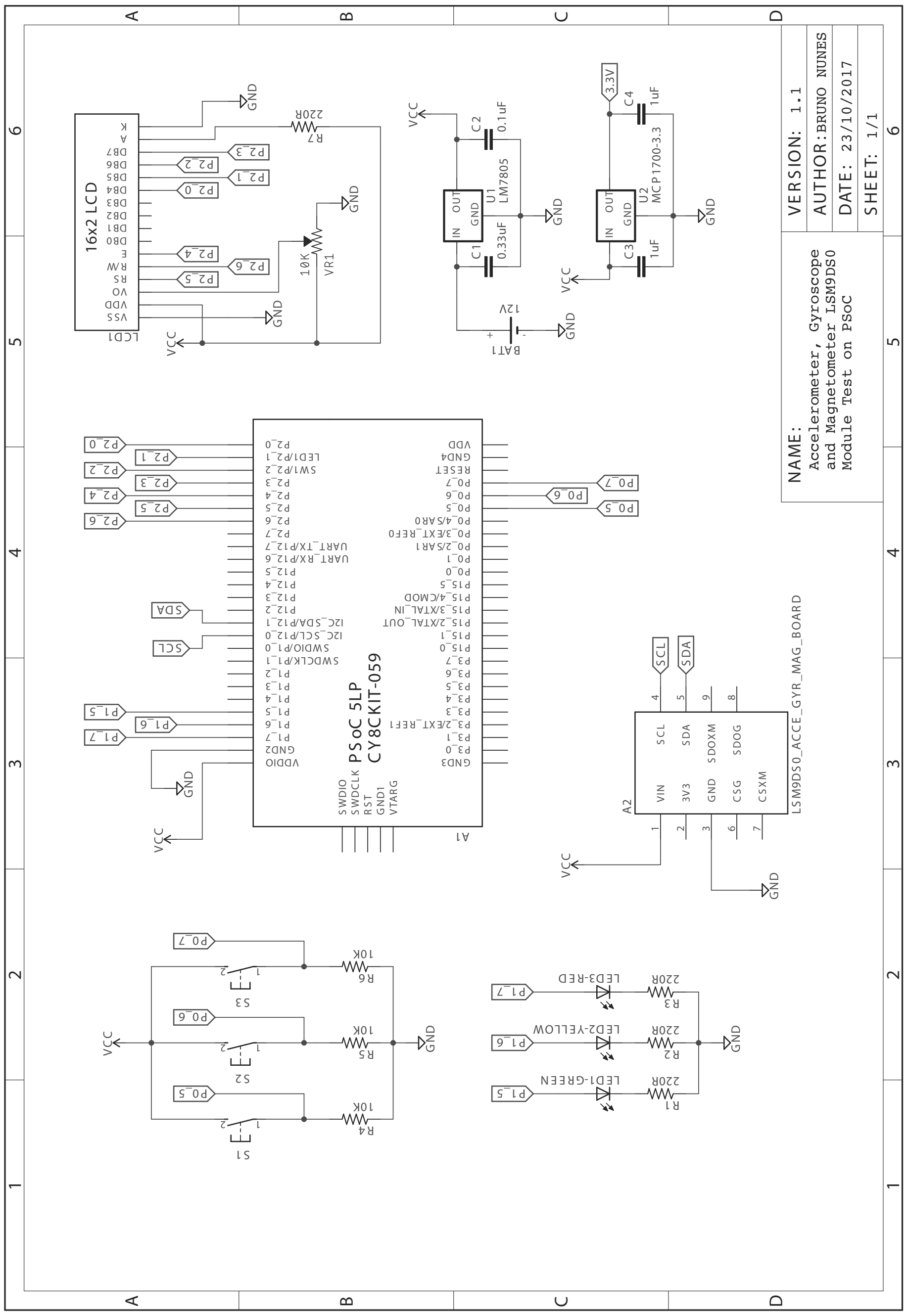


Figure IV.5: PSoC Magnetometer, Accelerometer and Gyroscope test schematic.

IV.5.2 Source Code

```
/* ========================================
 *
 *  PSoC
 *  MAGNETOMETER, ACCELEROMETER AND GYROSCOPE SENSOR TEST – Adafruit LSM9DS0
 *
 *  Created : 15/03/2018
 *  Modified: 09/02/2019
 *
 *  Version: 2.1
 * ========================================
*/
#include "project.h"
#include "stdio.h"
#include "string.h"

/* The I2C Slave address*/
#define LSM9DS0_ADDRESS_ACCELMAG      (0x1D)
#define LSM9DS0_ADDRESS_GYRO          (0x6B)

//Gyroscope
#define LSM9DS0_OUT_X_L_G_ADRESS    (0x28)
#define LSM9DS0_OUT_X_H_G_ADRESS    (0x29)
#define LSM9DS0_OUT_Y_L_G_ADRESS    (0x2A)
#define LSM9DS0_OUT_Y_H_G_ADRESS    (0x2B)
#define LSM9DS0_OUT_Z_L_G_ADRESS    (0x2C)
#define LSM9DS0_OUT_Z_H_G_ADRESS    (0x2D)
//Accelerometer
#define LSM9DS0_OUT_X_L_A_ADRESS    (0x28)
#define LSM9DS0_OUT_X_H_A_ADRESS    (0x29)
#define LSM9DS0_OUT_Y_L_A_ADRESS    (0x2A)
#define LSM9DS0_OUT_Y_H_A_ADRESS    (0x2B)
#define LSM9DS0_OUT_Z_L_A_ADRESS    (0x2C)
#define LSM9DS0_OUT_Z_H_A_ADRESS    (0x2D)
//Magnetometer
#define LSM9DS0_OUT_X_L_M_ADRESS    (0x08)
#define LSM9DS0_OUT_X_H_M_ADRESS    (0x09)
#define LSM9DS0_OUT_Y_L_M_ADRESS    (0x0A)
#define LSM9DS0_OUT_Y_H_M_ADRESS    (0x0B)
#define LSM9DS0_OUT_Z_L_M_ADRESS    (0x0C)
#define LSM9DS0_OUT_Z_H_M_ADRESS    (0x0D)

//Registers
#define LSM9DS0_CTRL_REG1_XM     (0x20)
#define LSM9DS0_CTRL_REG5_XM     (0x24)
#define LSM9DS0_CTRL_REG7_XM     (0x26)
#define LSM9DS0_CTRL_REG1_G      (0x20)

void get_data();
void data_analyser();
void init_LSM9DS0();

char str_gyro_x[15];
char str_gyro_y[15];
char str_gyro_z[15];
char str_accel_x[15];
char str_accel_y[15];
char str_accel_z[15];
char str_mag_x[15];
char str_mag_y[15];
char str_mag_z[15];

uint16_t gyro_buffer[6];
uint16_t accel_buffer[6];
uint16_t mag_buffer[6];

uint8_t state = 1;
uint8_t button_state = 0;


int main(void)
{
    CyGlobalIntEnable; /* Enable global interrupts. */

    /* Start the components */
  LCD1_Start();
    I2C1_Start();

    init_LSM9DS0();

    for(;;)
    {
        get_data();

        button_state = Button1_Read();

        if (button_state == 1)
            state++;

        if (state == 4)
            state = 1;

         /* Clear LCD */
        LCD1_ClearDisplay();

        if (state == 1)
        {
            LCD1_Position(0,0);
            LCD1_PrintString("Accel X: ");
            LCD1_Position(0,8);
            LCD1_PrintString(str_accel_x);
            LCD1_Position(1,0);
            LCD1_PrintString("Y: ");
            LCD1_Position(1,2);
            LCD1_PrintString(str_accel_y);
            LCD1_Position(1,9);
            LCD1_PrintString("Z: ");
            LCD1_Position(1,11);
            LCD1_PrintString(str_accel_z);
        }

        if (state == 2)
        {
            LCD1_Position(0,0);
            LCD1_PrintString("Mag X: ");
            LCD1_Position(0,8);
            LCD1_PrintString(str_mag_x);
            LCD1_Position(1,0);
            LCD1_PrintString("Y: ");
            LCD1_Position(1,2);
            LCD1_PrintString(str_mag_y);
            LCD1_Position(1,9);
            LCD1_PrintString("Z: ");
            LCD1_Position(1,11);
            LCD1_PrintString(str_mag_z);
        }

        if (state == 3)
        {
            LCD1_Position(0,0);
            LCD1_PrintString("Gyro X: ");
            LCD1_Position(0,8);
            LCD1_PrintString(str_gyro_x);
            LCD1_Position(1,0);
            LCD1_PrintString("Y: ");
            LCD1_Position(1,2);
            LCD1_PrintString(str_gyro_y);
            LCD1_Position(1,9);
            LCD1_PrintString("Z: ");
            LCD1_Position(1,11);
            LCD1_PrintString(str_gyro_z);
        }

        CyDelay(1000u);
    }
}

void init_LSM9DS0(){

    //setup Accel for continous mode
    I2C1_MasterSendStart(LSM9DS0_ADDRESS_ACCELMAG,I2C1_WRITE_XFER_MODE);
    I2C1_MasterWriteByte(LSM9DS0_CTRL_REG1_XM);
    I2C1_MasterWriteByte(0b01100111); // 100Hz data rate
    I2C1_MasterSendStop();
    I2C1_MasterSendStart(LSM9DS0_ADDRESS_ACCELMAG,I2C1_WRITE_XFER_MODE);
    I2C1_MasterWriteByte(LSM9DS0_CTRL_REG5_XM);
    I2C1_MasterWriteByte(0b11110000);
    I2C1_MasterSendStop();

    //setup Mag for continous mode
    I2C1_MasterSendStart(LSM9DS0_ADDRESS_ACCELMAG,I2C1_WRITE_XFER_MODE);
    I2C1_MasterWriteByte(LSM9DS0_CTRL_REG7_XM);
    I2C1_MasterWriteByte(0b00000000);
    I2C1_MasterSendStop();

    //setup Gyro for continous mode
    I2C1_MasterSendStart(LSM9DS0_ADDRESS_GYRO,I2C1_WRITE_XFER_MODE);
    I2C1_MasterWriteByte(LSM9DS0_CTRL_REG1_G);
    I2C1_MasterWriteByte(0b00001111);
    I2C1_MasterSendStop();
}

void get_data()
{
    int end = 1;

    while(end != 7)
    {
        //Initialize Transaction for Writing
        I2C1_MasterSendStart(LSM9DS0_ADDRESS_ACCELMAG,I2C1_WRITE_XFER_MODE);

        //Indicate Register to Write to
        switch(end)
        {
            case 1: I2C1_MasterWriteByte(LSM9DS0_OUT_X_L_A_ADRESS); break;
            case 2: I2C1_MasterWriteByte(LSM9DS0_OUT_X_H_A_ADRESS); break;
            case 3: I2C1_MasterWriteByte(LSM9DS0_OUT_Y_L_A_ADRESS); break;
            case 4: I2C1_MasterWriteByte(LSM9DS0_OUT_Y_H_A_ADRESS); break;
            case 5: I2C1_MasterWriteByte(LSM9DS0_OUT_Z_L_A_ADRESS); break;
            case 6: I2C1_MasterWriteByte(LSM9DS0_OUT_Z_H_A_ADRESS); break;
        }

        I2C1_MasterSendRestart(LSM9DS0_ADDRESS_ACCELMAG,I2C1_READ_XFER_MODE);

        //Read from Register (last byte NAKed)
        switch(end)
        {
            case 1: accel_buffer[0] = I2C1_MasterReadByte(I2C1_NAK_DATA);
    break;
```

```
                case 2: accel_buffer[1] = I2C1_MasterReadByte(I2C1_NAK_DATA);
    break;
                case 3: accel_buffer[2] = I2C1_MasterReadByte(I2C1_NAK_DATA);
    break;
                case 4: accel_buffer[3] = I2C1_MasterReadByte(I2C1_NAK_DATA);
    break;
                case 5: accel_buffer[4] = I2C1_MasterReadByte(I2C1_NAK_DATA);
    break;
                case 6: accel_buffer[5] = I2C1_MasterReadByte(I2C1_NAK_DATA);
    break;
            }

            //End Transaction
            I2C1_MasterSendStop();

            CyDelay(10);
            end++;
        }

        end = 1;


        while(end != 7)
        {
            //Initialize Transaction for Writing
            I2C1_MasterSendStart(LSM9DS0_ADDRESS_ACCELMAG,I2C1_WRITE_XFER_MODE);

            //Indicate Register to Write to
            switch(end)
            {
                case 1: I2C1_MasterWriteByte(LSM9DS0_OUT_X_L_M_ADRESS); break;
                case 2: I2C1_MasterWriteByte(LSM9DS0_OUT_X_H_M_ADRESS); break;
                case 3: I2C1_MasterWriteByte(LSM9DS0_OUT_Y_L_M_ADRESS); break;
                case 4: I2C1_MasterWriteByte(LSM9DS0_OUT_Y_H_M_ADRESS); break;
                case 5: I2C1_MasterWriteByte(LSM9DS0_OUT_Z_L_M_ADRESS); break;
                case 6: I2C1_MasterWriteByte(LSM9DS0_OUT_Z_H_M_ADRESS); break;
            }

            I2C1_MasterSendRestart(LSM9DS0_ADDRESS_ACCELMAG,I2C1_READ_XFER_MODE);

            //Read from Register (last byte NAKed)
            switch(end)
            {
                case 1: mag_buffer[0] = I2C1_MasterReadByte(I2C1_NAK_DATA); break;
                case 2: mag_buffer[1] = I2C1_MasterReadByte(I2C1_NAK_DATA); break;
                case 3: mag_buffer[2] = I2C1_MasterReadByte(I2C1_NAK_DATA); break;
                case 4: mag_buffer[3] = I2C1_MasterReadByte(I2C1_NAK_DATA); break;
                case 5: mag_buffer[4] = I2C1_MasterReadByte(I2C1_NAK_DATA); break;
                case 6: mag_buffer[5] = I2C1_MasterReadByte(I2C1_NAK_DATA); break;
            }

            //End Transaction
            I2C1_MasterSendStop();

            CyDelay(10);
            end++;
        }

        end = 1;

        while(end != 7)
        {
            //Initialize Transaction for Writing
            I2C1_MasterSendStart(LSM9DS0_ADDRESS_GYRO,I2C1_WRITE_XFER_MODE);

            //Indicate Register to Write to
            switch(end)
            {
                case 1: I2C1_MasterWriteByte(LSM9DS0_OUT_X_L_G_ADRESS); break;
                case 2: I2C1_MasterWriteByte(LSM9DS0_OUT_X_H_G_ADRESS); break;
                case 3: I2C1_MasterWriteByte(LSM9DS0_OUT_Y_L_G_ADRESS); break;
                case 4: I2C1_MasterWriteByte(LSM9DS0_OUT_Y_H_G_ADRESS); break;
                case 5: I2C1_MasterWriteByte(LSM9DS0_OUT_Z_L_G_ADRESS); break;
                case 6: I2C1_MasterWriteByte(LSM9DS0_OUT_Z_H_G_ADRESS); break;
            }

            I2C1_MasterSendRestart(LSM9DS0_ADDRESS_GYRO,I2C1_READ_XFER_MODE);

            //Read from Register (last byte NAKed)
            switch(end)
            {
                case 1: gyro_buffer[0] = I2C1_MasterReadByte(I2C1_NAK_DATA); break;
                case 2: gyro_buffer[1] = I2C1_MasterReadByte(I2C1_NAK_DATA); break;
                case 3: gyro_buffer[2] = I2C1_MasterReadByte(I2C1_NAK_DATA); break;
                case 4: gyro_buffer[3] = I2C1_MasterReadByte(I2C1_NAK_DATA); break;
                case 5: gyro_buffer[4] = I2C1_MasterReadByte(I2C1_NAK_DATA); break;
                case 6: gyro_buffer[5] = I2C1_MasterReadByte(I2C1_NAK_DATA); break;
            }

            //End Transaction
            I2C1_MasterSendStop();

            CyDelay(10);
            end++;
        }

        data_analyser();
    }

    void data_analyser()
    {
        uint8_t accel_x_low, accel_y_low, accel_z_low;
        int16_t accel_x_high, accel_y_high, accel_z_high;
        uint8_t mag_x_low, mag_y_low, mag_z_low;
        int16_t mag_x_high, mag_y_high, mag_z_high;
        uint8_t gyro_x_low, gyro_y_low, gyro_z_low;
        int16_t gyro_x_high, gyro_y_high, gyro_z_high;
        float accel_x, accel_y, accel_z;
        float mag_x, mag_y, mag_z;
        float gyro_x, gyro_y, gyro_z;

        accel_x_low = accel_buffer[0];
        accel_x_high = accel_buffer[1];
        accel_y_low = accel_buffer[2];
        accel_y_high = accel_buffer[3];
        accel_z_low = accel_buffer[4];
        accel_z_high = accel_buffer[5];

        accel_x_high <<= 8; //shift 8 bytes to the left
        accel_x_high |= accel_x_low; //concatenate low byte to high byte
        accel_y_high <<= 8; //shift 8 bytes to the left
        accel_y_high |= accel_y_low; //concatenate low byte to high byte
        accel_z_high <<= 8; //shift 8 bytes to the left
        accel_z_high |= accel_z_low; //concatenate low byte to high byte
        accel_x = ((accel_x_high*0.061F)/1000)*9.80665F;
        accel_y = ((accel_y_high*0.061F)/1000)*9.80665F;
        accel_z = ((accel_z_high*0.061F)/1000)*9.80665F;
        snprintf(str_accel_x, sizeof(str_accel_x), "%.2f", accel_x);
        snprintf(str_accel_y, sizeof(str_accel_y), "%.2f", accel_y);
        snprintf(str_accel_z, sizeof(str_accel_z), "%.2f", accel_z);

        mag_x_low = mag_buffer[0];
        mag_x_high = mag_buffer[1];
        mag_y_low = mag_buffer[2];
        mag_y_high = mag_buffer[3];
        mag_z_low = mag_buffer[4];
        mag_z_high = mag_buffer[5];

        mag_x_high <<= 8; //shift 8 bytes to the left
        mag_x_high |= mag_x_low; //concatenate low byte to high byte
        mag_y_high <<= 8; //shift 8 bytes to the left
        mag_y_high |= mag_y_low; //concatenate low byte to high byte
        mag_z_high <<= 8; //shift 8 bytes to the left
        mag_z_high |= mag_z_low; //concatenate low byte to high byte
        mag_x = (mag_x_high*0.08F)/1000;
        mag_y = (mag_y_high*0.08F)/1000;
        mag_z = (mag_z_high*0.08F)/1000;
        snprintf(str_mag_x, sizeof(str_mag_x), "%.2f", mag_x);
        snprintf(str_mag_y, sizeof(str_mag_y), "%.2f", mag_y);
        snprintf(str_mag_z, sizeof(str_mag_z), "%.2f", mag_z);

        gyro_x_low = gyro_buffer[0];
        gyro_x_high = gyro_buffer[1];
        gyro_y_low = gyro_buffer[2];
        gyro_y_high = gyro_buffer[3];
        gyro_z_low = gyro_buffer[4];
        gyro_z_high = gyro_buffer[5];

        gyro_x_high <<= 8; //shift 8 bytes to the left
        gyro_x_high |= gyro_x_low; //concatenate low byte to high byte
        gyro_y_high <<= 8; //shift 8 bytes to the left
        gyro_y_high |= gyro_y_low; //concatenate low byte to high byte
        gyro_z_high <<= 8; //shift 8 bytes to the left
        gyro_z_high |= gyro_z_low; //concatenate low byte to high byte
        gyro_x = gyro_x_high*0.00875F;
        gyro_y = gyro_y_high*0.00875F;
        gyro_z = gyro_z_high*0.00875F;
        snprintf(str_gyro_x, sizeof(str_gyro_x), "%.2f", gyro_x);
        snprintf(str_gyro_y, sizeof(str_gyro_y), "%.2f", gyro_y);
        snprintf(str_gyro_z, sizeof(str_gyro_z), "%.2f", gyro_z);

    }
    /* [] END OF FILE */
```

Listing IV.4: Arduino Accelerometer, Magnetometer and Gyroscope Test Code

IV.6 RTC

IV.6.1 Schematic

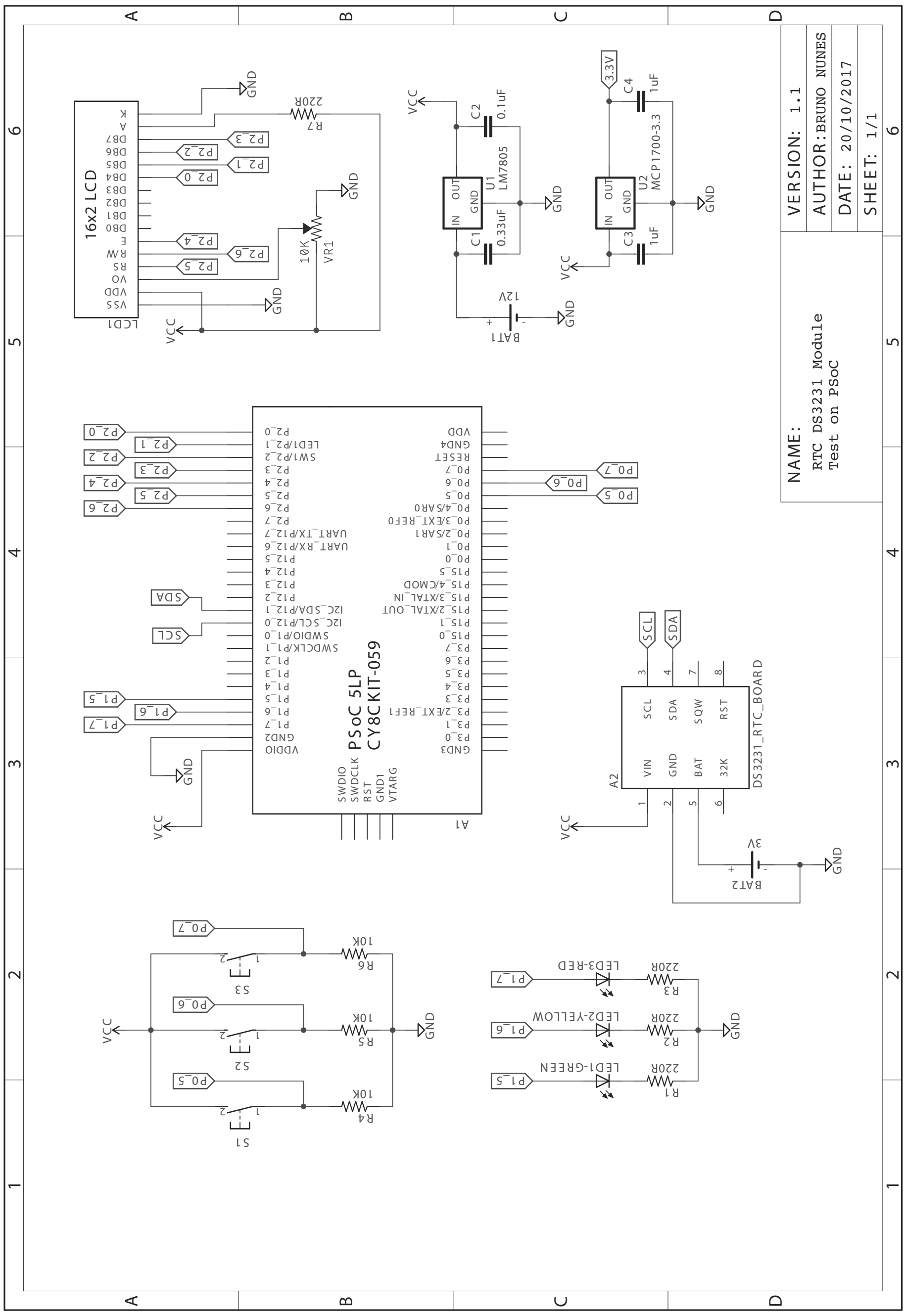


Figure IV.6: PSoC RTC test schematic.

IV.6.2 Source Code

```c
/* ========================================
 *
 *  PSoC
 *  RTC TEST - Adafruit DS3231
 *
 *  Created : 21/02/2018
 *  Modified: 03/02/2019
 *
 *  Version: 2.1
 * ========================================
*/
#include "project.h"
#include "stdio.h"
#include "string.h"
#include "math.h"
/*In Creator go to Project -> Build Settings -> Linker -> General
Add "m" (without the quotes) into Additional Libraries.
*/

/* The I2C Slave address*/
#define DS3231_ADDRESS      (0x68)

#define DS3231_SECONDS_ADRESS    (0x00)
#define DS3231_MINUTES_ADRESS    (0x01)
#define DS3231_HOURS_ADRESS      (0x02)
#define DS3231_DAY_ADRESS        (0x04)
#define DS3231_MONTH_ADRESS      (0x05)
#define DS3231_YEAR_ADRESS       (0x06)

void get_time();
void configure_time();
void data_analyser();
int convertBinaryToDecimal(long n);

uint8_t buffer_seconds;
uint8_t buffer_minutes;
uint8_t buffer_hours;
uint8_t buffer_day;
uint8_t buffer_month;
uint8_t buffer_year;

char str_day[15];
char str_month[15];
char str_year[15];
char str_hours[15];
char str_minutes[15];
char str_seconds[15];

```

```
int main(void)
{
    CyGlobalIntEnable; /* Enable global interrupts. */

    /* Start the components */
  LCD1_Start();
    I2C1_Start();

    //configure_time(); //run only once

    for(;;)
    {
        get_time();

        /* Clear LCD */
        LCD1_ClearDisplay();

        LCD1_Position(0,0);
        LCD1_PrintString("Date: ");
        LCD1_Position(0,6);
        LCD1_PrintString(str_day);
        LCD1_Position(0,8);
        LCD1_PrintString("/");
        LCD1_Position(0,9);
        LCD1_PrintString(str_month);
        LCD1_Position(0,11);
        LCD1_PrintString("/");
        LCD1_Position(0,12);
        LCD1_PrintString(str_year);


        LCD1_Position(1,0);
        LCD1_PrintString("Time:");
        LCD1_Position(1,6);
        LCD1_PrintString(str_hours);
        LCD1_Position(1,8);
        LCD1_PrintString(":");
        LCD1_Position(1,9);
        LCD1_PrintString(str_minutes);
        LCD1_Position(1,11);
        LCD1_PrintString(":");
        LCD1_Position(1,12);
        LCD1_PrintString(str_seconds);

        CyDelay(1000u);
    }
}

void get_time()
{
```

```c
    int end = 1;

    while(end != 7)
    {
        //Initialize Transaction for Writing
        I2C1_MasterSendStart(DS3231_ADDRESS,I2C1_WRITE_XFER_MODE);

        //Indicate Register to Write to
        switch(end)
        {
            case 1: I2C1_MasterWriteByte(DS3231_SECONDS_ADRESS); break;
            case 2: I2C1_MasterWriteByte(DS3231_MINUTES_ADRESS); break;
            case 3: I2C1_MasterWriteByte(DS3231_HOURS_ADRESS); break;
            case 4: I2C1_MasterWriteByte(DS3231_DAY_ADRESS); break;
            case 5: I2C1_MasterWriteByte(DS3231_MONTH_ADRESS); break;
            case 6: I2C1_MasterWriteByte(DS3231_YEAR_ADRESS); break;
        }

        I2C1_MasterSendRestart(DS3231_ADDRESS,I2C1_READ_XFER_MODE);

        //Read from Register (2 Bytes, last byte NAKed)
        switch(end)
        {
            case 1: buffer_seconds = I2C1_MasterReadByte(I2C1_NAK_DATA); break
    ;
            case 2: buffer_minutes = I2C1_MasterReadByte(I2C1_NAK_DATA); break
    ;
            case 3: buffer_hours = I2C1_MasterReadByte(I2C1_NAK_DATA); break;
            case 4: buffer_day = I2C1_MasterReadByte(I2C1_NAK_DATA); break;
            case 5: buffer_month = I2C1_MasterReadByte(I2C1_NAK_DATA); break;
            case 6: buffer_year = I2C1_MasterReadByte(I2C1_NAK_DATA); break;
        }

        //End Transaction
        I2C1_MasterSendStop();

        CyDelay(10);
        end++;
    }

    data_analyser();
}

void configure_time()
{
    int end = 1;

    //Time will be configure to 03/02/2019 16:36:0

    uint8_t config_seconds = 0b00000000; //0
    uint8_t config_minutes = 0b00110110; //36
    uint8_t config_hours = 0b00010110; //16
    uint8_t config_day = 0b00000011; //3
    uint8_t config_month = 0b00000010; //2
    uint8_t config_year = 0b00011001; //19

    while(end != 7)
    {
        //Initialize Transaction for Writing
        I2C1_MasterSendStart(DS3231_ADDRESS,I2C1_WRITE_XFER_MODE);

        //Indicate Register to Write to
        switch(end)
        {
            case 1: I2C1_MasterWriteByte(DS3231_SECONDS_ADRESS); break;
            case 2: I2C1_MasterWriteByte(DS3231_MINUTES_ADRESS); break;
            case 3: I2C1_MasterWriteByte(DS3231_HOURS_ADRESS); break;
            case 4: I2C1_MasterWriteByte(DS3231_DAY_ADRESS); break;
            case 5: I2C1_MasterWriteByte(DS3231_MONTH_ADRESS); break;
            case 6: I2C1_MasterWriteByte(DS3231_YEAR_ADRESS); break;
        }

        //Write in BCD format
        switch(end)
        {
            case 1: I2C1_MasterWriteByte(config_seconds); break;
            case 2: I2C1_MasterWriteByte(config_minutes); break;
            case 3: I2C1_MasterWriteByte(config_hours); break;
            case 4: I2C1_MasterWriteByte(config_day); break;
            case 5: I2C1_MasterWriteByte(config_month); break;
            case 6: I2C1_MasterWriteByte(config_year); break;
        }

        //End Transaction
        I2C1_MasterSendStop();

        CyDelay(10);
        end++;
    }


}

void data_analyser()
{
    uint8_t buffer_seconds_copy, seconds_dec1, seconds_dec2, seconds;
    uint8_t buffer_minutes_copy, minutes_dec1, minutes_dec2, minutes;
    uint8_t buffer_hours_copy, hours_dec1, hours_dec2, hours;
    uint8_t buffer_day_copy, day_dec1, day_dec2, day;
    uint8_t buffer_month_copy, month_dec1, month_dec2, month;
    uint8_t buffer_year_copy, year_dec1, year_dec2;
    uint16_t year;

    buffer_seconds_copy = buffer_seconds;
    buffer_seconds &= 0xF; //clears bit 4, 5, 6 , 7
    buffer_seconds_copy &= 0x70; //clears bit 0, 1, 2, 3 and bit 7
    buffer_seconds_copy >>= 4; //shift 4 bits to the right
    seconds_dec1 = convertBinaryToDecimal(buffer_seconds);
    seconds_dec2 = convertBinaryToDecimal(buffer_seconds_copy);
    seconds = ( seconds_dec2 * 10 ) + seconds_dec1;
    snprintf(str_seconds, sizeof(str_seconds), "%d", seconds);

    buffer_minutes_copy = buffer_minutes;
    buffer_minutes &= 0xF; //clears bit 4, 5, 6 , 7
    buffer_minutes_copy &= 0x70; //clears bit 0, 1, 2, 3 and bit 7
    buffer_minutes_copy >>= 4; //shift 4 bits to the right
    minutes_dec1 = convertBinaryToDecimal(buffer_minutes);
    minutes_dec2 = convertBinaryToDecimal(buffer_minutes_copy);
    minutes = ( minutes_dec2 * 10 ) + minutes_dec1;
    snprintf(str_minutes, sizeof(str_minutes), "%d", minutes);

    buffer_hours_copy = buffer_hours;
    buffer_hours &= 0xF; //clears bit 4, 5, 6 , 7
    buffer_hours_copy &= 0x30; //clears bit 0, 1, 2, 3 and bit 6, 7
    buffer_hours_copy >>= 4; //shift 4 bits to the right
    hours_dec1 = convertBinaryToDecimal(buffer_hours);
    hours_dec2 = convertBinaryToDecimal(buffer_hours_copy);
    hours = ( hours_dec2 * 10 ) + hours_dec1;
    snprintf(str_hours, sizeof(str_hours), "%d", hours);

    buffer_day_copy = buffer_day;
    buffer_day &= 0xF; //clears bit 4, 5, 6 , 7
    buffer_day_copy &= 0x30; //clears bit 0, 1, 2, 3 and bit 6, 7
    buffer_day_copy >>= 4; //shift 4 bits to the right
    day_dec1 = convertBinaryToDecimal(buffer_day);
    day_dec2 = convertBinaryToDecimal(buffer_day_copy);
    day = ( day_dec2 * 10 ) + day_dec1;
    snprintf(str_day, sizeof(str_day), "%d", day);

    buffer_month_copy = buffer_month;
    buffer_month &= 0xF; //clears bit 4, 5, 6 , 7
    buffer_month_copy &= 0x10; //clears bit 0, 1, 2, 3 and bit 6, 7
    buffer_month_copy >>= 4; //shift 4 bits to the right
    month_dec1 = convertBinaryToDecimal(buffer_month);
    month_dec2 = convertBinaryToDecimal(buffer_month_copy);
    month = ( month_dec2 * 10 ) + month_dec1;
    snprintf(str_month, sizeof(str_month), "%d", month);

    buffer_year_copy = buffer_year;
    buffer_year &= 0xF; //clears bit 4, 5, 6 , 7
    buffer_year_copy &= 0xF0; //clears bit 0, 1, 2, 3
    buffer_year_copy >>= 4; //shift 4 bits to the right
    year_dec1 = convertBinaryToDecimal(buffer_year);
    year_dec2 = convertBinaryToDecimal(buffer_year_copy);
    year = 2000 + ( year_dec2 * 10 ) + year_dec1;
    snprintf(str_year, sizeof(str_year), "%d", year);
}

int convertBinaryToDecimal(long n)
{
    int decimal = 0;
    int i = 0;
    int remainder;

    while (n!=0)
    {
        remainder = n  %10;
        n /= 10;
        decimal += remainder*pow(2,i);
        ++i;
    }
    return decimal;
}

/* [] END OF FILE */
```

Listing IV.5: Arduino RTC Test Code

Annex V

Other Payload Tests - Technical Files

V.1 PSOC and Arduino Communication

V.1.1 Schematic

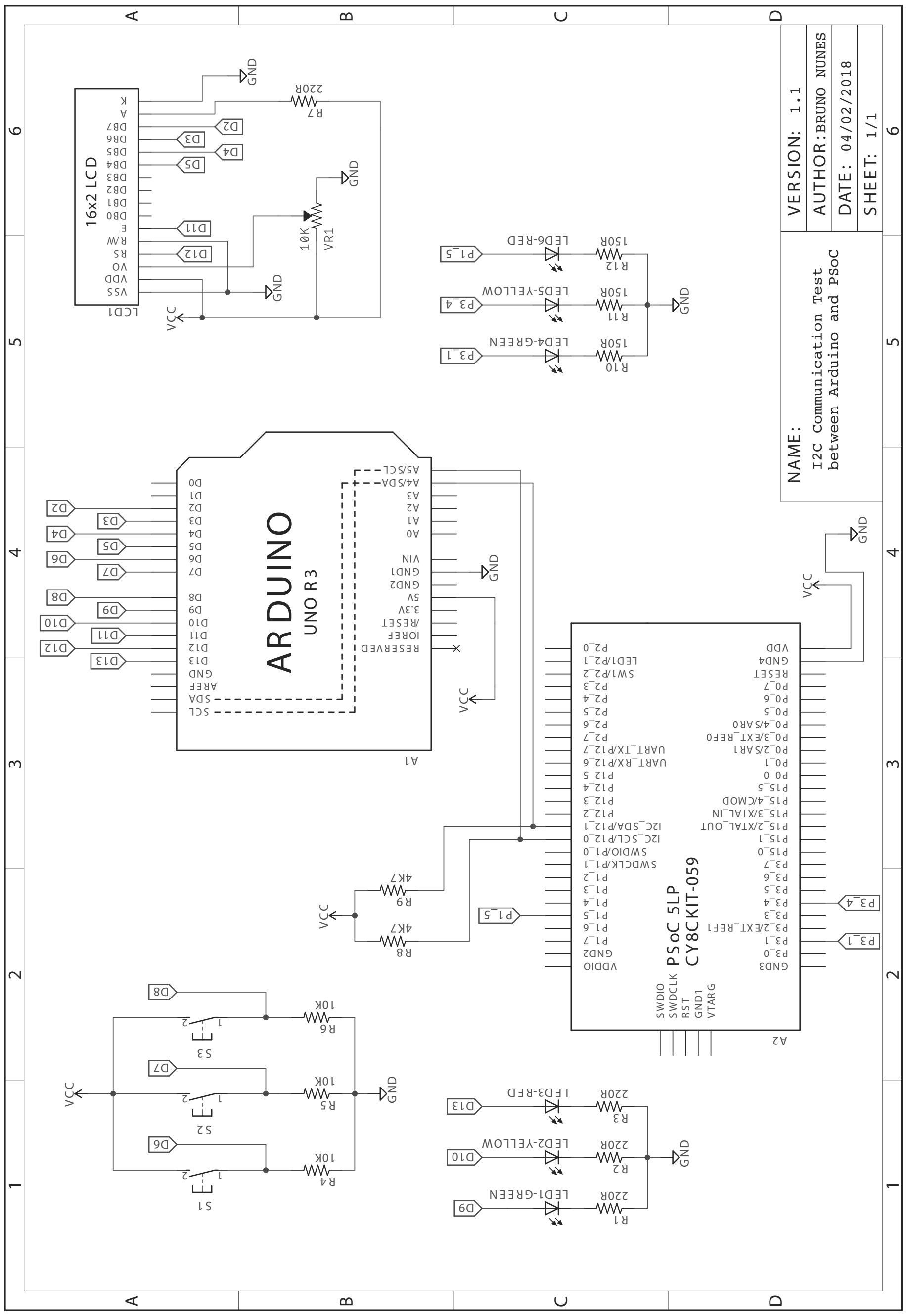


Figure V.1: Arduino <-> PSoC I2C communication test schematic.

V.1.2 Source Code

V.1.2.1 Arduino

```
/*
 *  I2C Communication TEST - Arduino - PSoC
 *
 *  Created : 16/01/2018
 *  Modified: 20/02/2019
 *
 *  Version: 1.5
 *
 */

#include <LiquidCrystal.h>
#include <Wire.h>
#include <TimerOne.h>

int B1_pin = 6;
int B2_pin = 7;
int B3_pin = 8;

int LED_Green_pin = 9; //receive led
int LED_Yellow_pin = 10;
int LED_Red_pin = 13; //send led

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

int led_red_state = 0;             // led state used to set the LED
int led_green_state = 0;

unsigned long currentMillis;

unsigned long previous_red_Millis = 0;        // will store last time LED was
    updated
unsigned long previous_green_Millis = 0;
unsigned long previous_comm_Millis = 0;
unsigned long previous_init_comm_Millis = 0;

const long interval = 3000;   //Led ON time
const long interval_comm = 15000;  //delay between 1st and 2nd transmission
const long interval_init_comm = 5000; //initial communciation delay

int state = 0; //state 1 - send request; state 2 - receive data; state 3 -
    send request; state 4 - receive data

char req_confirm[3];

char datain1[32];
char datain2[32];
```

```

int i = 0;

void setup() {

  lcd.begin(16, 2); //(columns, rows)

  pinMode (B1_pin, INPUT);
  pinMode (B2_pin, INPUT);
  pinMode (B3_pin, INPUT);

  pinMode (LED_Green_pin, OUTPUT);
  pinMode (LED_Yellow_pin, OUTPUT);
  pinMode (LED_Red_pin, OUTPUT);

  Wire.begin();
  Serial.begin(9600);

  Timer1.initialize(1000); //1 ms
  Timer1.attachInterrupt(time_interrupt);

  lcd.clear();

}

void time_interrupt(){

  currentMillis = millis();

  if (currentMillis - previous_red_Millis >= interval) {
    previous_red_Millis = currentMillis; //save the last time
    if (led_red_state == 1) { //Turn the Led Off
      digitalWrite(LED_Red_pin,LOW);
      led_red_state = 0;
    }
  }

  if (currentMillis - previous_green_Millis >= interval) {
    previous_green_Millis = currentMillis; //save the last time
    if (led_green_state == 1) { //Turn the Led Off
      digitalWrite(LED_Green_pin,LOW);
      led_green_state = 0;
    }
  }

  if (currentMillis - previous_comm_Millis >= interval_comm) {
    previous_comm_Millis = currentMillis; //save the last time
    if (state == 5) { //send next data request
      state = 3;
    }
  }

  if (currentMillis - previous_init_comm_Millis >= interval_init_comm) {
    previous_init_comm_Millis = currentMillis; //save the last time
    if (state == 0) { //start all again
      state = 1;
    }
  }
}

void loop() {

  if (state == 1 || state == 3){

    for (i = 0; i <= strlen(req_confirm); i++)
      req_confirm[i]=0;

    Wire.beginTransmission(8); // transmit to device #8
    if(state == 1)
      Wire.write("NeedData");
    if(state == 3)
      Wire.write("MoreData");
    Wire.endTransmission (); // stop transmitting

    if(state == 1)
      Serial.println("Arduino Request 1: NeedData");
    if(state == 3)
      Serial.println("Arduino Request 2: MoreData");

    Wire.requestFrom(8, 2);    // request 2 bytes from slave device #8
    i = 0;

    while (Wire.available()) { // slave may send less than requested
      char c = Wire.read(); // receive a byte as character
      req_confirm[i] = c;
      i++;
    }

    if (req_confirm[0]=='o' && req_confirm[1]=='k'){
      digitalWrite (LED_Red_pin, HIGH);
      led_red_state = 1;
      previous_red_Millis = currentMillis;
      Serial.print("PSoC Answer: ");
      Serial.print(req_confirm);
      Serial.println("");
    }

    if(state == 1)
      state = 2;
    if(state == 3)
```

```
      state = 4;
  }


  if(state == 2 || state == 4){

    for(i = 0; i <= strlen(datain1); i++)
      datain1[i]=0;
    for(i = 0; i <= strlen(datain2); i++)
      datain2[i]=0;

    Wire.requestFrom(8, 32);     // request 32 bytes from slave device #8
    i = 0;

    while (Wire.available()) { // slave may send less than requested
      char c = Wire.read(); // receive a byte as character
      if(state == 2)
        datain1[i] = c;
      if(state == 4)
        datain2[i] = c;
      i++;
    }

    if (datain1[0] == '#' || datain2[0] == '#'){
      digitalWrite (LED_Green_pin, HIGH);
      led_green_state = 1;
      previous_green_Millis = currentMillis;

      Serial.print("Data Received from PSoC: ");
      if(state == 2){
        Serial.print(datain1);
        state = 5;
      }
      if(state == 4){
        Serial.print(datain2);
        state = 0;
        previous_init_comm_Millis = currentMillis;
      }
      Serial.println("");
    }

    lcd.setCursor(0, 0);
    if(state == 2){
      lcd.print("Data 1 Received");
    }
    if(state == 4){
      lcd.print("Data 2 Received");
    }

    Wire.beginTransmission(8); // transmit to device #8
    Wire.write("ok");
    Wire.endTransmission (); // stop

}
  delay(200);
}
```

Listing V.1: Arduino -> PSoC I2C Communication Test Code

V.1.2.2 PSoC

```
/* ========================================
 *
 *  PSoC
 *  I2C Communication TEST - Arduino - PSoC
 *
 *  Created : 19/01/2018
 *  Modified: 23/02/2019
 *
 *  Version: 1.9
 * ========================================
*/
#include "project.h"
#include "stdio.h"
#include "stdlib.h"
#include "string.h"

void data_collector();
CY_ISR(MY_ISR);
void sendData();

/* Set the write buffer length to be 32 bites or 4 bytes */
#define RD_BUFFER_SIZE          (32u)

uint16 ms_count = 0;
uint16 ms_count2 = 0;
uint16 ms_count3 = 0;
uint16 ms_count4 = 0;

uint16 sensor1 = 0u; //'u' to signify that they are unsigned meaning strictly
    positive integers.
uint16 sensor2 = 0u;
uint16 sensor3 = 0u;
uint16 sensor4 = 0u;
float sensor5 = 3.85;

char sample_data[RD_BUFFER_SIZE];
char sample_data2[RD_BUFFER_SIZE];

```

```c
char str[32];
char str2[32];

int red = 0;
int green = 0;
int nsends = 0;

int state = 1; //state 1 - receive request 1; state 2 - send data1; state 3 - request 2; state 4 - send data2; state 5 - transmission confirmation
int req = 0; //request

int sendconfirm = 0;
char confirm[] = "ok";

int main(void)
{
    Timer0_Start(); // Configure and enable timer
    isr1_StartEx(MY_ISR); // Point to MY_ISR to carry out the interrupt sub-routine
    CyGlobalIntEnable; /* Enable global interrupts. */

    for (uint i = 0; i <= RD_BUFFER_SIZE; i++)
        str[i] = 0;

    for (uint i = 0; i <= RD_BUFFER_SIZE; i++)
        str2[i] = 0;

    char i2c_receive[9];
    /* Set up slave write data buffer */
    I2C0_SlaveInitWriteBuf((uint8 *) i2c_receive, 9);

    char i2c_send[RD_BUFFER_SIZE];
    /* Set up slave read data buffer */
    I2C0_SlaveInitReadBuf((uint8 *) i2c_send, RD_BUFFER_SIZE);

    I2C0_Start();

    for(;;)
    {
        if(I2C0_SlaveStatus() & I2C0_SSTAT_WR_CMPLT){
            I2C0_SlaveClearWriteBuf();
            I2C0_SlaveClearWriteStatus();
        }

        if(state == 1 || state == 3)
        {
            if(i2c_receive[0]=='N' && i2c_receive[1]=='e' && i2c_receive[2]=='e' && i2c_receive[3]=='d'
            && i2c_receive[4]=='D' && i2c_receive[5]=='a' && i2c_receive[6]=='t' && i2c_receive[7]=='a'
            && state == 1)
            {
                Led_Green_Write(1u);
                ms_count2 = 0; // reset ms counter
                state = 5;
                req = 1;
            }

            if(i2c_receive[0]=='M' && i2c_receive[1]=='o' && i2c_receive[2]=='r' && i2c_receive[3]=='e'
            && i2c_receive[4]=='D' && i2c_receive[5]=='a' && i2c_receive[6]=='t' && i2c_receive[7]=='a'
            && state == 3)
            {
                Led_Green_Write(1u);
                ms_count2 = 0; // reset ms counter
                state = 5;
                req = 2;
            }
        }

        if(state == 2 || state == 4)
        {
            for (uint i = 0;i<=RD_BUFFER_SIZE;i++)
                i2c_send[i] = 0;

            for (uint i = 0;i<=RD_BUFFER_SIZE;i++)
                i2c_receive[i] = 0;

            if(state == 2)
                strcpy(i2c_send,str);
            if(state == 4)
                strcpy(i2c_send,str2);

            while(!sendconfirm)
            {
                sendData();

                if(I2C0_SlaveStatus() & I2C0_SSTAT_WR_CMPLT){
                    I2C0_SlaveClearWriteBuf();
                    I2C0_SlaveClearWriteStatus();

                    if(i2c_receive[0]=='o' && i2c_receive[1]=='k'){
                        sendconfirm=1;
                        Led_Red_Write(1u);
                        ms_count = 0; // reset ms counter
                        if(state == 2)
                            state = 3;
                        if(state == 4)
                            state = 1;
                    }
                }
            }
            sendconfirm=0;
        }

        if(state == 5)
        {
            for (uint i = 0;i<=RD_BUFFER_SIZE;i++)
                i2c_send[i] = 0;

            strcpy(i2c_send,confirm);

            nsends=0;
            ms_count4=0;
            while(!nsends)
                sendData();

            Led_Red_Write(1u);
            ms_count = 0; // reset ms counter
            if(req==1)
                state = 2;
            if(req==2)
                state = 4;
        }
    }
}

void data_collector()
{
    sensor1 += 2;
    sensor2 += 5;
    sensor3 = 576;
    sensor4 += 3;
    sensor5 += 0.5;

    /* Float to string:
        - set Heap Size (bytes) to 0x200 in System
        - Project -> Build Settings -> ARM GCC -> Linker:
            Set use newlib-nano Float Formatting to True
    */
    snprintf(str, sizeof(str), "#%d,%d,%d#", sensor1, sensor2, sensor3);
    snprintf(str2, sizeof(str2), "#%d,%.2f#", sensor4, sensor5);
}
void sendData()
{
     /* Check if the slave buffer has been read */
    if(I2C0_SlaveStatus() & I2C0_SSTAT_RD_CMPLT)
    {
        I2C0_SlaveClearReadStatus();
        I2C0_SlaveClearReadBuf();
    }
}
CY_ISR(MY_ISR)
{
    ms_count++;
    if(ms_count == 3000) // 3 second
        Led_Red_Write(0u); // Toggle LED

    ms_count2++;
    if(ms_count2 == 3000) // 3 second
        Led_Green_Write(0u); // Toggle LED

    ms_count3++;
    if(ms_count3 == 1000) // 1 second
    {
        data_collector();
        ms_count3 = 0;
    }

    ms_count4++;
    if(ms_count4 == 500) // 0.5 second
        nsends = 1;
}

/* [] END OF FILE */
```

Listing V.2: PSoC -> Arduino I2C Communication Test Code

V.2 Voltage Step-up and Battery Charger

V.2.1 Schematic

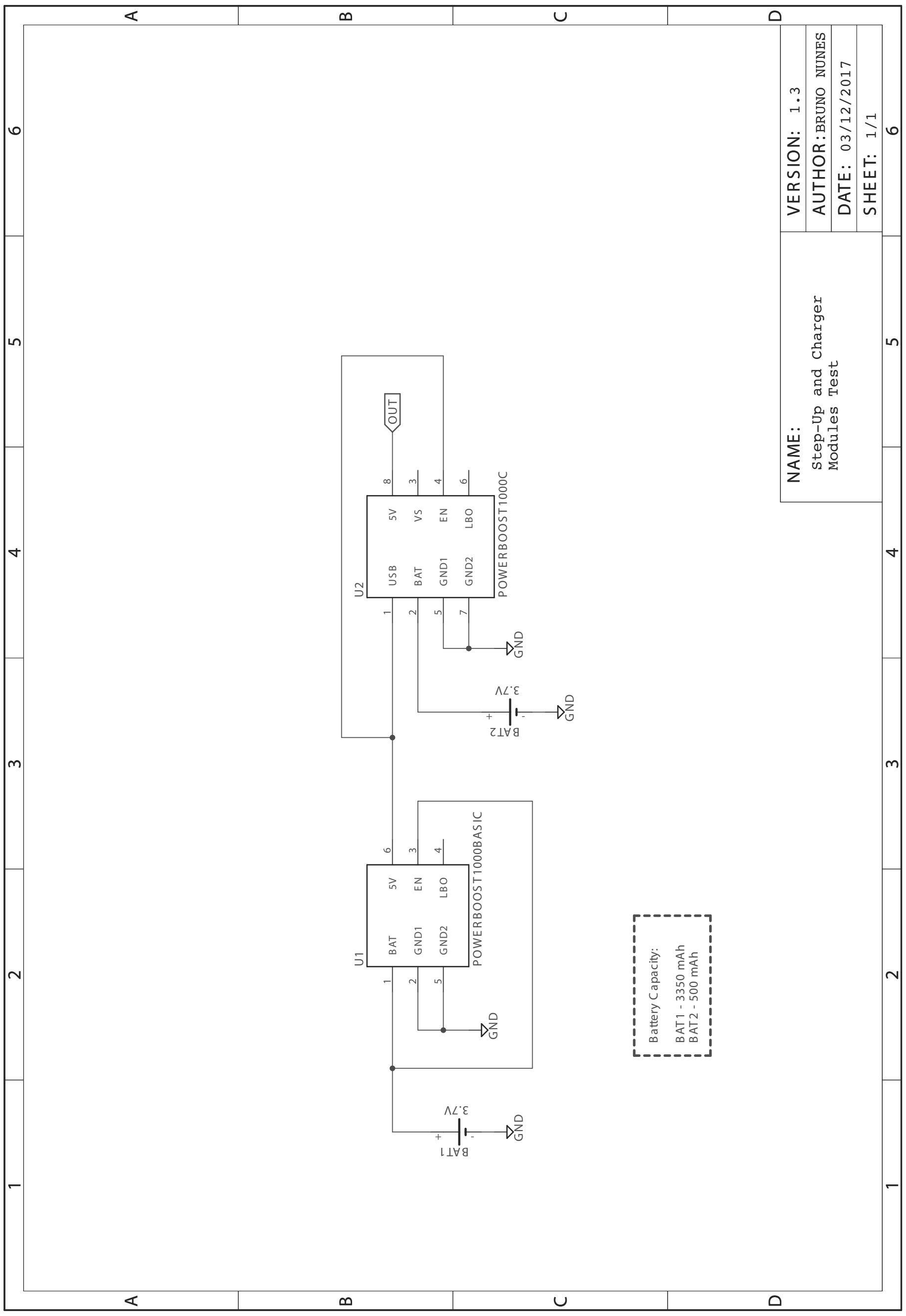


Figure V.2: Voltage step-up and battery charger test schematic.

Annex VI

Payload Board Tests - Technical Files

VI.1 Schematic

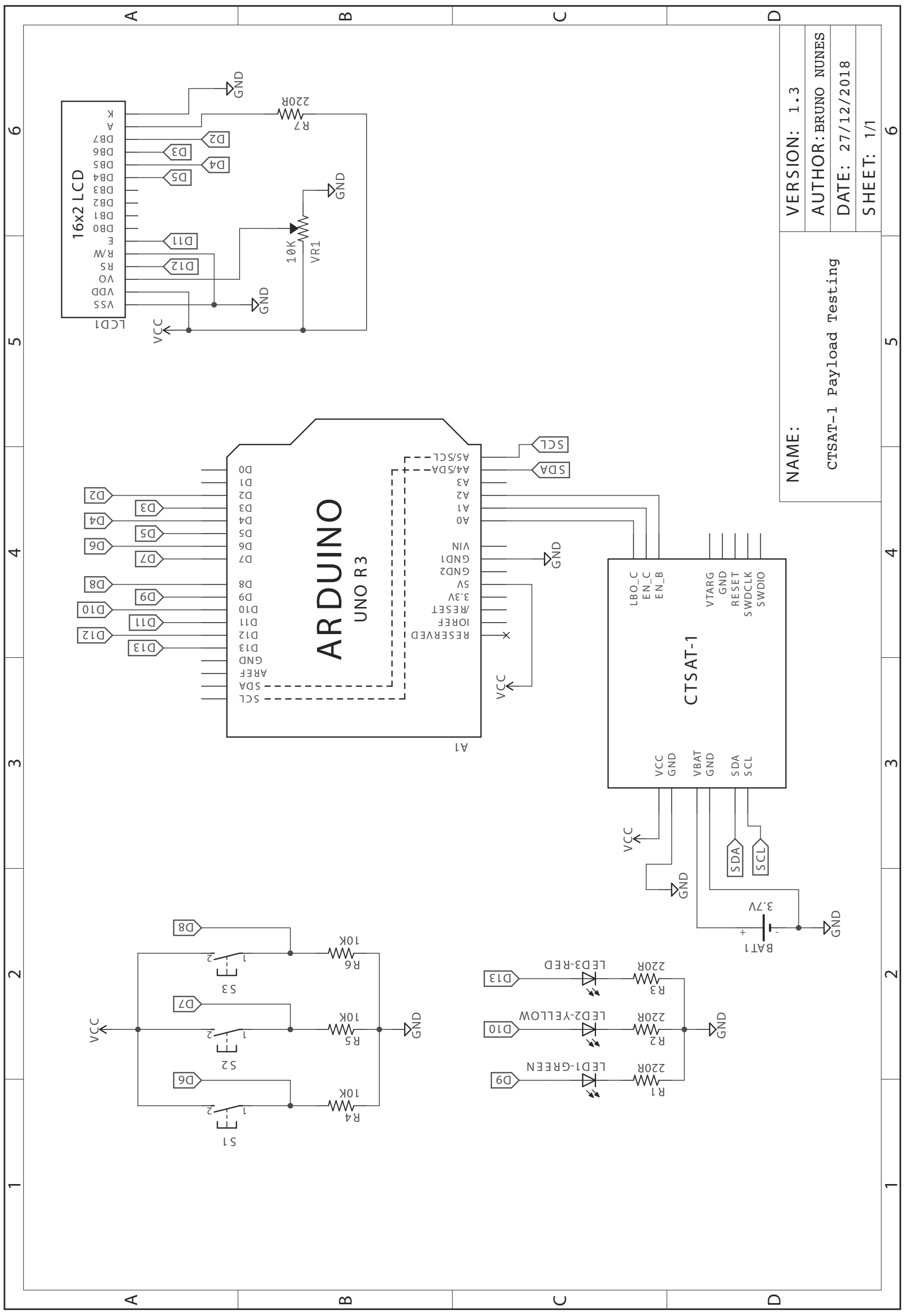


Figure VI.1: CTSAT-1 <-> Arduino test schematic.